IBM General Information Manual

305 RAMAC® Programmer's Guide

CONTENTS

INTRODUCTION

One constantly sought after goal of a machine installation is increased speed and accuracy in the accomplishment of its assigned task.

To obtain these optimum results with the IBM 305 RAMAC, certain programming features must be considered in the utilization of its components. Effective use of all stored program and control panel features, proper use of the disk file and good operating techniques all play a major role in producing an efficient program.

This manual is intended to serve as a guide for developing detailed machine programs and operating techniques. It should be used as a starting point only, since many applications may require the development of new methods and routines. Programmers should be guided by the circumstances within each application. The 305 RAMAC is a powerful tool, and every avenue should be explored to take full advantage of the facilities provided.

PROGRAM PLANNING

Information is normally processed in the 305 by a continuous-flow type operation. Thus, a transaction enters the system via the card reader and, after processing, generally results in an output document being produced. It is obvious, then, that no one area of the system can process more items than another. Also, the over-all transaction processing speed will be determined by the speed of the slowest area. Therefore, for maximum operating efficiency, all areas of processing should be balanced as closely as possible.

The areas listed in this section are separated as a convenience for study and program analysis. Each of the areas has a real effect on the others; therefore, they must all be considered and planned as one, rather than as a series of isolated steps. There is a natural tendency to disregard output considerations until after the rest of the program has been planned. This is a serious mistake in a continuous-process system and may become a bottleneck for the entire operation. The 305 machine program, from source document to machine output must be considered as a whole. A proper balance in all of these areas will assure maximum efficiency of the over-all system operation.

The process to develop a program is basic to one set of control panels and stored instructions. The same general procedure should be followed for each program consisting of separate control panels and stored instructions. For example, in a complex billing application, it may be desirable to process Receipts and Issues with one program and other types of transactions with a separate program. Although each program should be treated separately, decisions made in one program should be continuously related to the other to arrive at the best compatible arrangement.

INPUT CONSIDERATIONS

Data input to the 305 is by IBM punched cards. The read brushes sense the punched holes from the back of the card; therefore, sense marking on the front of the card will not affect the accuracy of punched-hole reading. However, sense marking on the back of the cards may cause erroneous card reading and must not be used.

Each card is read and checked internally before it is made available to the processing unit via track K. Card columns 1-80 are recorded on track positions 00-79. Track positions 80-99 (inclusive) are cleared to blanks on each card reading cycle. The principle function of the read check mechanism is to stop the machine if the two independent sets of brushes interpret a card differently. However, any double-punched card column that uses a character

"bit" more than once in the combined punchings will be detected as an error. For instance, a card column double-punched 9-5 will signal an error because a "1" bit is used in both the "9" and "5". A card column double-punched 6-1 will not cause an error signal because no bit is common to both digit values. The bit structure for all characters used in the RAMAC is shown in Figure 1.

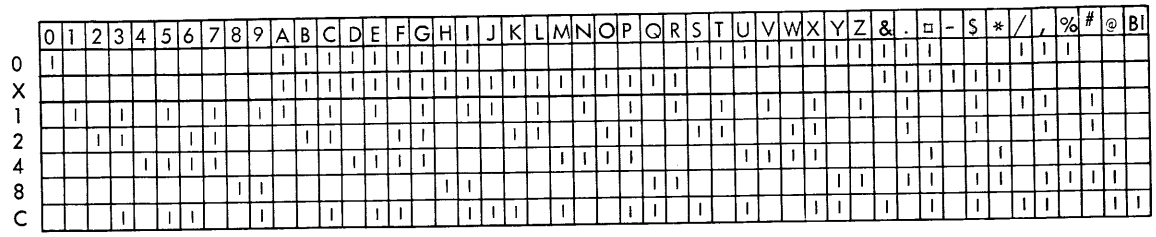


Figure 1

From a systems standpoint, two types of cards may be used for input.

1. Single-item (or transaction) cards.
2. Multi-item cards (usually referred to as spread-item cards).

The single-item card, as the name implies, contains information pertaining to only one item or transaction. It may be produced automatically by various methods. When this is done, it may be desirable to punch the same information in several fields of the card. The format on the card should be arranged to facilitate the transfer of data within the 305.

The spread-item card is used when the amount of information required for each item is limited (Figure 2).

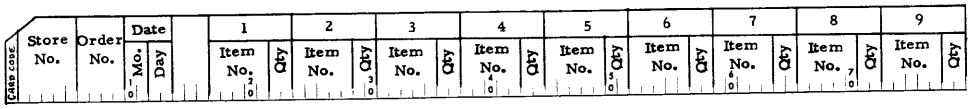


Figure 2. Spread-Item Card

The main advantage of the spread-item card is that it reduces card volume. However, single-item cards are easier to incorporate in audit trails and in any processing that may be necessary with unit record machines. Program restart routines usually are more difficult with spread-item cards.

The program for a spread-item card usually is written so that the data in item position 1 of the card (Figure 2) is processed first, and then a track-slide

operation is performed to move items 2-9 one field to the left. The slide instruction for Figure 2 would be K86 K79 63 A6. Note that, as item 9 is moved into the item-8 position, the item-9 position is blanked. Thus, as each slide operation is performed, another item position on the track is blanked. After the 9th slide operation, all 9-item positions will be blank and the blank transmission selector will be transferred. Program exit A, of the slide instruction, tests for blank transmission on each slide operation, and at the appropriate time causes a new card to be read.

To minimize card-read time,some programs have employed a "stacking" technique. This involves reading several cards into the machine on a series of tracks. Each time a card is used off the end, the card data tracks are shifted and a new card is read from track K. This technique, in some cases, can save card-read time. However, it places an undue burden upon the operator whenever a restart routine is required. For this reason, it is not recommended except in very unusual situations.

All data entered into the RAMAC should be verified for correctness. There are many ways of providing this verification, such as the self-checking number device for part numbers, the proof punch for accumulation of control totals, accounting machine balancing, pre-punching, and complete verification with the various types of verifiers. This is of major importance since many RAMAC applications will involve the updating of multiple records with input data, and if the basic data is incorrect, it is immediately reflected in many records. The process of correcting these errors is both time-consuming and costly, and can nearly always be avoided through the use of proper verification methods.

DISK FILE CONSIDERATIONS

Planning

The disk file is the heart of the RAMAC. It provides the means to store extensive records and allows each record to be available for use almost immediately. In order for this facility to be utilized most efficiently for any given application the programmer must select a suitable file arrangement. Among the factors to be considered are:

1. the number of different types of records,
2. the number of records within each type,
3. the number of characters in the records of each type,
4. the physical distribution or arrangement of records within the file, and
5. the method of addressing the records.

Information pertaining to items 1, 2, and 3 is used to determine the total file storage requirement. It may sometimes develop that there is more information to be stored in the file than there is storage capacity. Further study might disclose that it would not be economical to keep all this data in the file, even if there were capacity. For instance, one RAMAC program seemed to require file storage for 25,000 customer records alone. It was later determined that only 10 percent of these customers were active more than once per year. Similar situations have been found with many items in inventory control. When this occurs, it is best to store the most active items in the file and handle the remainder on an exception basis. This can be done by either pre-analyzing the card and preventing it from entering the machine if it refers to an item not in file, or constructing a program which recognizes this situation and signals the condition by punching a card or typing an appropriate message. If there is the possibility of an input card referring to an item not in file, provision must be made for this condition.

Another factor which must be considered in planning file space, is that of future expansion. If future requirements cannot be determined realistically, a minimum of 10 percent should be left for this purpose.

Although the file is divided into 50,000 one-hundred character sectors, each directly addressable by a five-digit number, there is nothing to prevent the utilization of longer or shorter records. For instance, if the record length were 200 characters, then two 100-character locations would be required for each 200-character record. The capacity of the file would be 25,000 records of this type. If direct addressing were used for item numbers 00000 through 24999, the even numbered file addresses could be used for the first 100 characters and the next higher address for the second 100 characters of the record. An item number 13493 would be multiplied by two to develop file address 26986, and its next higher location 26987. Item 13494 would occupy address locations 26988, 26989, and so forth.

Another method of locating records greater than 100 characters is to seek the first file location of the record by its address and record advance for the remaining locations. If the record is not stored in consecutive locations, each location can contain, along with its part of the record, the address of the location of the next part of that record.

Short records may be handled in several ways. If two different applications each require 50-character records, then the file locations may be directly addressed on each program. The same record address would be used in both routines. One routine would be programmed to use one end of the record, and the other program would use the opposite end. Both programs would read and write 100 characters in the file. Another variation is to use some equal division of the record track, and use a sixth digit in the item identification to indicate

track position. For instance, item 123453 would locate 12345. The sixth digit (3) would be used to modify the processing programs so as to refer to the third division of the record. An example of this method is shown on page 43. Combinations of long and short records can be devised to answer a particular need.

Addressing Methods

Considerable time and effort have been spent on developing methods that will yield maximum efficiency in addressing the disk storage of the 305 RAMAC. To achieve this maximum efficiency, data must be stored, and subsequently retrieved, in a manner that minimizes the average time required to locate an item in storage, and also minimizes the amount of unused (but assigned) storage space. The ideal addressing method, therefore, will yield a file that utilizes 100 percent of the disk storage area allotted to it and in which each item can be located with only one seek.

Several different methods of disk file addressing have been developed. Each of these methods has certain advantages and disadvantages which vary in importance according to the nature of the data composing the file. The methods may be divided into two general categories, direct addressing and indirect addressing.

Direct Addressing

In using the direct addressing method, certain information in each record is used, either directly or by a simple conversion process, to provide the disk file address. This information is usually the control data of the record (i. e., an identifying part of the record such as employee number, part number, or account number). This control data most frequently will reflect a change or addition that was made in order to facilitate disk file addressing, but could be an original part of the record.

For example: A file consists of 3,000 customer records; each record can be contained in one disk sector, and the account numbers of the records have been modified to run consecutively from 20,000 to 22,999. If desired, the account numbers can be used directly to locate the file in disk sectors 20000 to 22999. Or, if it is desired to locate the records in another 3,000 consecutive disk sectors, a simple addition or subtraction process will provide the desired addresses. In either case, an ideal assignment of addresses will result because all items can be located with only one seek, and the disk file area utilized will be full.

The advantages obtainable by use of direct addressing are:

1. 100 percent utilization of file storage space can be attained.
2. Access time is minimized as each item can be located with only one seek.
3. No machine time is required to convert the control data of the record to a file address.
4. No cross-index is required for inquiry.
5. File maintenance (additions and deletions) can be easily handled.

From a processing viewpoint, therefore, direct addressing is the most efficient method of disk file addressing and should be used whenever practical. The practicality of using this method can be determined by weighing the benefits derived from the advantages, just enumerated, against the cost of changing the control data to conform with direct addressing.

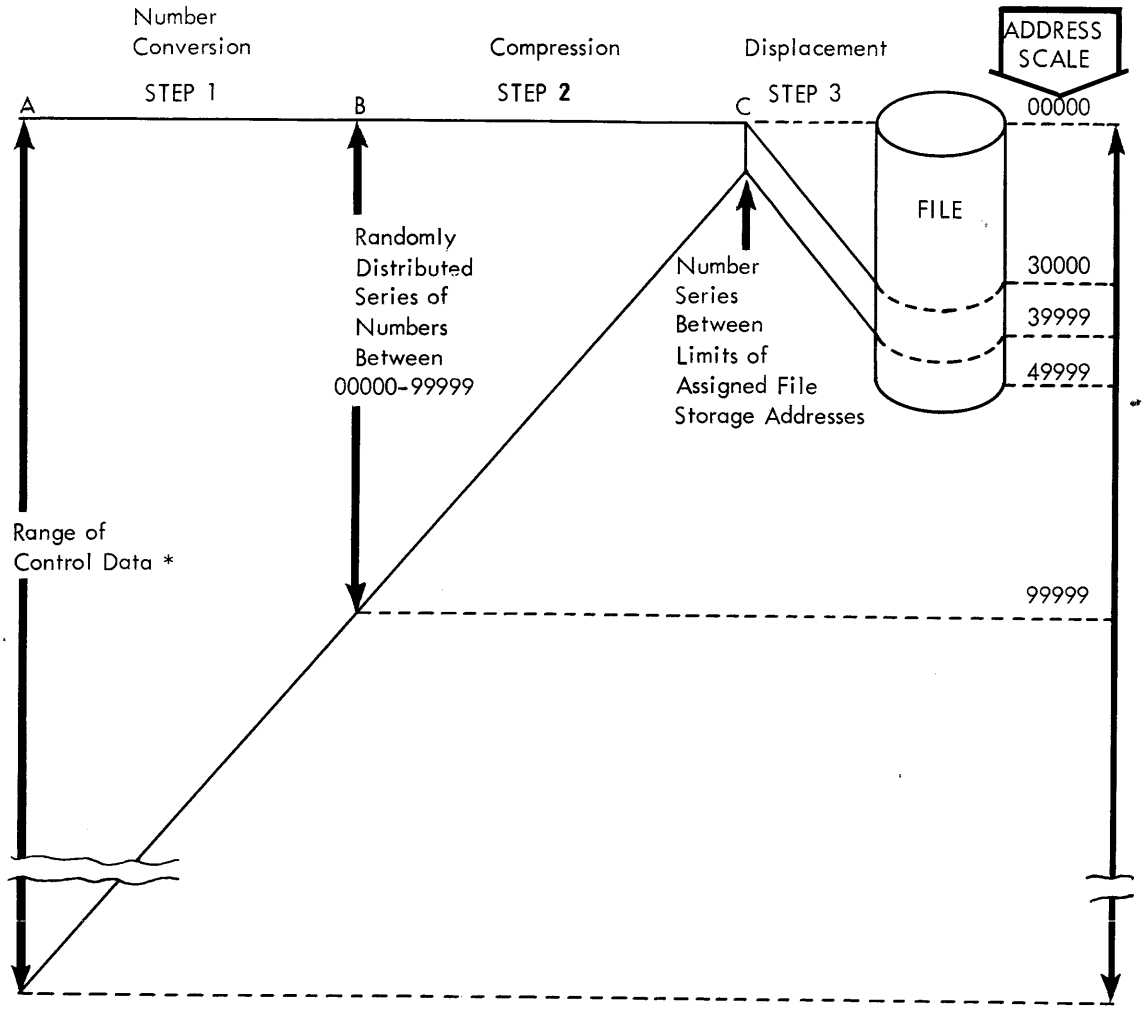
Indirect Addressing

In some applications, it is not practical to use direct addressing. These situations usually arise because no part of the records will directly be, or easily convert to, disk file addresses. Also, there will be a few instances in which direct addressing will not be possible because the available addresses, if used, will result in an excessive amount of wasted file space. In either case, it will be necessary to use indirect addressing methods.

In developing an efficient indirect addressing system, the basic issues to be resolved are address conversion and disk file addressing. In address conversion the control data must be operated on by a procedure to provide an address within the desired range. This procedure, which is illustrated in Figure 3, consists basically of the following three steps:

Step 1. An arithmetic process must be devised which will, when applied to all control data, produce a suitably random distribution of numbers in a given range. This is usually 00000-99999 when converting control data to five-digit addresses and 0000-9999 when converting to four-digit addresses.

Step 2. The number series produced in Step 1 must be compressed to fit the desired number of record locations. For example, if control data is converted to five-digit addresses and only 10,000 addresses are to be used, the compression factor would be .1. This reduces the number series to a range from 00000 to 09999. (NOTE: In some procedures, it may be possible to combine Step 1 and Step 2.)



* Between the limits of the numbers in use, there usually is a very large number of possibilities. However, only a small percentage of these numbers is actually used. These numbers are the ones to be used later in the number conversion process.

Figure 3

Step 3. If necessary, a displacement constant is now added to the number series. This displacement constant is equal to the first address of the file area to which the items are assigned. For example, 30,000 could be added to the number series 00000 to 09999 to develop disk file addresses 30000 to 39999.

The illustration shows that address conversion is like a funnel directing data into the file. The example illustrates the use of 10,000 file sectors to store data from a list of identification numbers (control data) with possibilities far exceeding this number. However, in the example, it should have been determined that the maximum number of actual sectors required was approximately 8500. In planning a conversion method, it must be determined how many numbers are used in a system having a given number of possibilities.

Experience has shown that the first step in planning an address conversion process is to develop a complete knowledge of the control data. One point to remember here is that the zone portion of alphabetic characters is not used in the 305 arithmetic circuits, except in the sign-control positions. Therefore, the study should be made on the basis of the numerical portion of the control data only. One approach to this study is to develop a distribution of the existing numbers between the two extremes (lowest to highest). This distribution will illustrate the requirements of the address conversion process. Another approach is to develop statistics relating to the control data. A count can be made to determine digit distribution within each number position. Also, the frequency of "pairing" of digits between columns should be considered. For instance, the distribution of individual digit values in each of two columns may be nearly even, but most of the 3's in one column may coincide with the 6's in the other to form a constant of 36. Thus, there will be few 16's, 26's, 46's, 35's, 37's, and so forth. For these counts, IBM unit record equipment can be used.

Usually, while developing a knowledge of the control data, certain facts will stand out. It may be found that one position is predominantely zero, or that another position distributes evenly. From these facts a conversion method, consisting of selecting five digits from the control data, may be chosen and evaluated as explained on page 14. However, it may be determined that complete address conversion is required. If so, there are many different techniques that can be employed. A few of the simpler techniques, as shown in Section 3, of the Appendix to this manual, should be tried first, since they frequently will produce satisfactory results. It should be noted that, if a complicated formula is devised to produce near-perfect results, it will usually be extremely sensitive to control data changes. A small number of additions and deletions may have a major effect on its performance.

There is no common formula which will produce uniformly good results in all cases. For each application, a formula must be selected which is best suited to the conditions of the particular situation. Several conversion processes should be tried and evaluated before deciding on the final process to be used.

Any standard IBM calculator can be used to compute the converted addresses in the same manner as would be done by the 305. One card is punched with the control data for each item to be stored in the file. These cards are processed through the calculator and the computed file address punched in each card. Some calculators may even compute the converted address for several different formulae on one pass of the cards.

In developing an address conversion procedure, the over-all objective is the production of addresses which will aid in achieving maximum efficiency in 305 RAMAC disk file addressing. Five factors should be considered in this context: overflows, item activity, file packing, file maintenance, and machine processing time. The latter four factors may be considered in their relationship to overflows.

Items which are located in home addresses (the original address developed in converting control data) may be retrieved with only one seek; overflows (items which are not located in the original developed address) require more than one seek. It is obvious, therefore, that minimizing the references to overflows will minimize the average seek time required to locate each record. This is accomplished by minimizing the number of overflows, and by locating those items which are referred to most frequently in, or as near as possible to, home addresses; i. e., minimizing overflow items, which are among the more active items of the file. Thus, file activity is an important factor to be considered in developing an addressing procedure. If the expected activity of all items in the file is approximately equal, then file activity will have little influence on average seek time per record.

However, if file activity is unequal (e. g., 20 percent of the items account for 70 percent of the total transaction activity), it will have a considerable influence on average seek time. In this latter case, seek time will be minimized if the more active items are located in, or as close as possible to, their respective home addresses. The conversion technique should therefore be designed, primarily, for performance on the high-activity items, and those items should be loaded into the file first, thus giving them priority locations. If several seek operations are required to locate an item referred to only once a week, it will have little over-all effect.

Disk file packing should also be considered when developing an address conversion procedure. Usually, as packing density increases, the number of overflow items increases (thereby increasing the average seek time per record) and it becomes more and more difficult to develop a satisfactory conversion procedure.

Therefore, records should be packed as loosely as possible in the disk storage area.

The effects of subsequent file maintenance should, if possible, be estimated. As mentioned earlier in this section, a complicated formula yielding near-perfect results will usually be extremely sensitive to control number changes. A small number of additions and deletions may have a major effect on its performance.

Finally, the machine processing time required to convert control data to disk file addresses should be considered.

Address conversion procedures can be evaluated, in terms of average seek time per record, by using the methods described in the 305 RAMAC Bulletin, "The Chaining Method of Disk File Addressing for the 305 RAMAC," form J28-2008. If the converted addresses are to be used in a "chaining" process, an evaluation may be made exactly as described in the bulletin. Addresses developed for use in an "Indexing" process can be evaluated by pretending that they are for chaining and following the procedures described in the bulletin. The number of seeks per home address required to locate all records in the chain should be developed as indicated in Figure 2 of the bulletin. For example, if nine records can be stored on each track, the following chart would be developed.

References per Address	9 Records per Address	References per Address	9 Records per Address	References per Address	9 Records per Address
1	1	8	8	15	36
2	2	9	9	16	44
3	3	10	11	17	53
4	4	11	14	18	63
5	5	12	18	19	74
6	6	13	23	20	86
7	7	14	29	etc.	---

Whenever an indirect addressing method is used, it is necessary to maintain a cross-index reference for manual inquiry purposes. For each item loaded in the file, a card should be punched containing the control data and actual file address of the item. These cards can be used to print the cross-index reference. It may be wise, in some applications, to avoid using file address 00000 for storage of valuable data. This address is sought if all zeros, all blanks, or any combination of zeros and blanks are sent to the address register.

Because any address conversion procedure will develop a varying number of duplicate addresses (synonyms), methods must be provided for locating those records (overflows) that are not in the original address developed (the home address). The remaining portion of this section is devoted to a discussion of these methods. An outline of basic concepts and concerns that pertain to any and all methods is presented first. This is followed by discussion of three specific methods: Next Available Address, Indexing, and Chaining. The discussion of each specific method parallels the outline of basic concepts and concerns.

Basic Concepts and Concerns

1. Address Capacity--How many items can be located in each address?

2. Nature of Items in Each Address--Is each item an actual record, or is it control data which will be used to locate an actual record?

3. Retrieval of Overflow Items--How will overflow items be retrieved from disk storage? One technique for accomplishing this requires an address search procedure; another requires specific overflow addresses stored at the home location and each successive overflow location. These techniques function as follows:

    a. Address Search Procedure--In this technique, the home address is examined to see if it contains the desired item. If it does not, a set pattern of searching other addresses (usually, but not necessarily, the next higher sequential address) is used until the overflow item is found. An example of an address search procedure is included in the discussion of the "next available address" method.

    b. Specific Overflow Address--In this technique, as in the preceding one, the home address is examined to see if it contains the desired item. If it does not, an "overflow address," which was placed in the home address at the time the first overflow occurred, is sent to the address register, and the overflow location is checked to see if it contains the desired item. If it does not, a second overflow address, which was placed in the first overflow location when the second overflow occurred, is sent to the address register, etc. This process continues until the desired item is found.

4. Loading the File--Provision must be made for loading items into disk storage. In indirect addressing, this is complicated by the existence of

overflow items. Both the storage and retrieval of these items must be taken into consideration in developing a loading program. Further information concerning these factors is included in the discussions of specific methods.

5. Deleting Items from the File--

6. Adding Items to the File-- } Provision must also be made for deleting items from the file and adding items to the file. Deletions and additions may be accomplished by various techniques, some of which are described in the following discussions of specific methods.

The Next Available Address Method

Although this method is adequate for some applications, it often will prove to be less efficient because too great a number of seeks will be required, on the average, to find each record. It is included in this manual primarily for illustrative purposes, and only secondarily as a method to be considered for use in an actual application.

1. Address Capacity--Usually, but not necessarily, one item may be located in each address.

2. Nature of Items in Each Address--Each item is an actual record.

3. Retrieval of Overflow Items--Overflow items are retrieved from disk storage by means of an "address search procedure." If the desired item is not found in the home address, the next higher addresses in the disk file are searched sequentially until the desired item is found or the upper limit of the file area is reached. In the latter case, the file area is then searched starting at the lower limit until the item is not in the file. Figure 4 illustrates this type of address search.

4. Loading the File--The disk file area to be used should be blanked before loading. In this method, the file is loaded in one pass. Each record that cannot be stored in a home address is placed in the next available higher address. This is done by means of a sequential search similar to the one used in retrieving records, except that an address is sought in which the record can be stored. As in any indirect addressing technique, records should be loaded according to expected activity, i. e., the most active items should be loaded first and the least active items last.

5. Deleting Items from the File--Deletions are made by searching for the item to be deleted, and then blanking out the entire item.

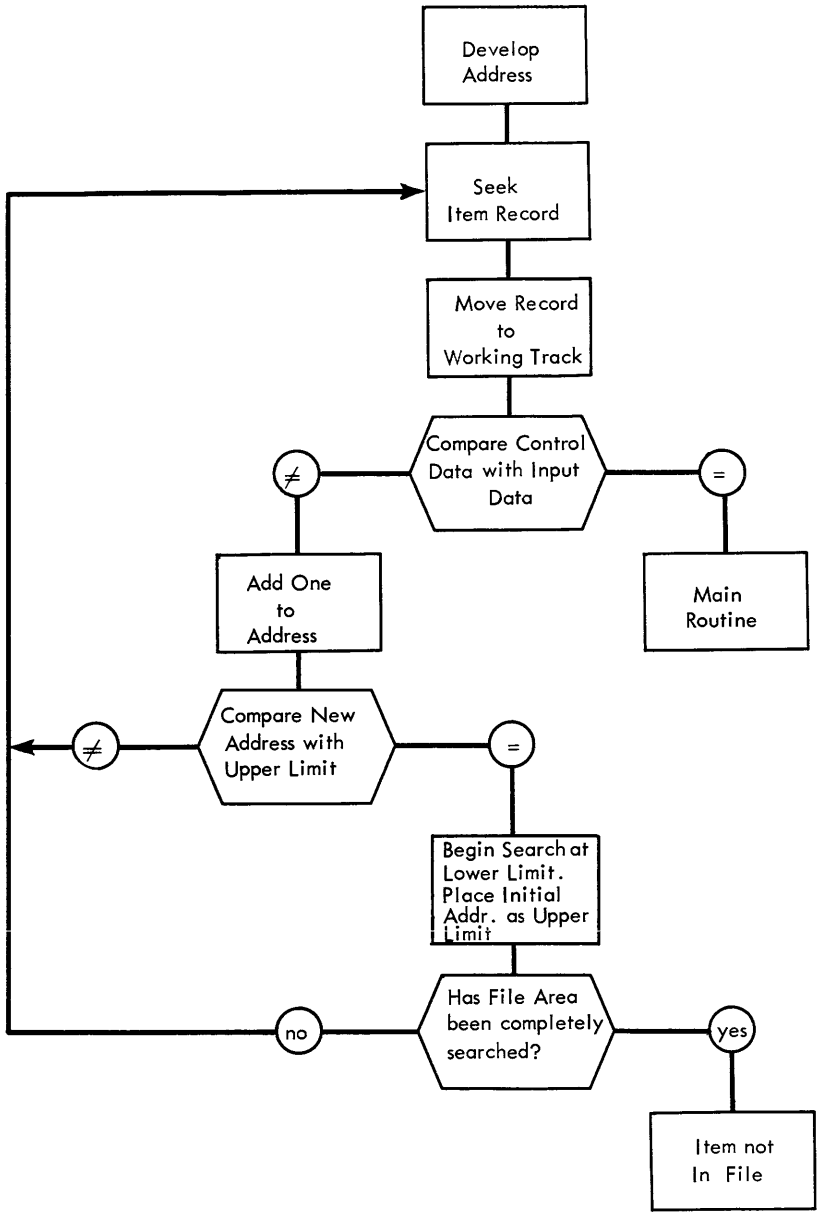


Figure 4

6. Adding Items to File--Items are added to the file in the same manner as the file is loaded, i. e., the home address is sought and, if empty, the item is placed there. If occupied, a sequential search is used to find the next available address in which the item can be stored.

Indexing

1. Address Capacity--Usually, but not necessarily, nine items may be located in each address.

2. Nature of Items in Each Address--Each item is control data that is used to locate an actual record. These control data items are arranged as an "index record" on the zero sector of each track in the same order as the corresponding records are arranged on the disk sectors of the track. This permits use of the built-in field compare and skip-to-record features of the 305 to retrieve all records located on the track.

3. Retrieval of Overflow Items--Overflow items are retrieved from disk storage by means of a "specific overflow address" procedure. Each zero sector includes space in which an overflow address can be placed. This address specifies the next zero sector to be examined for the desired item.

   Thus, when the desired item is not found in the home address, each overflow location which may contain the item is examined sequentially until the item is found, or it is determined that the item is not in the file. Figure 5 illustrates an indexing retrieval process.

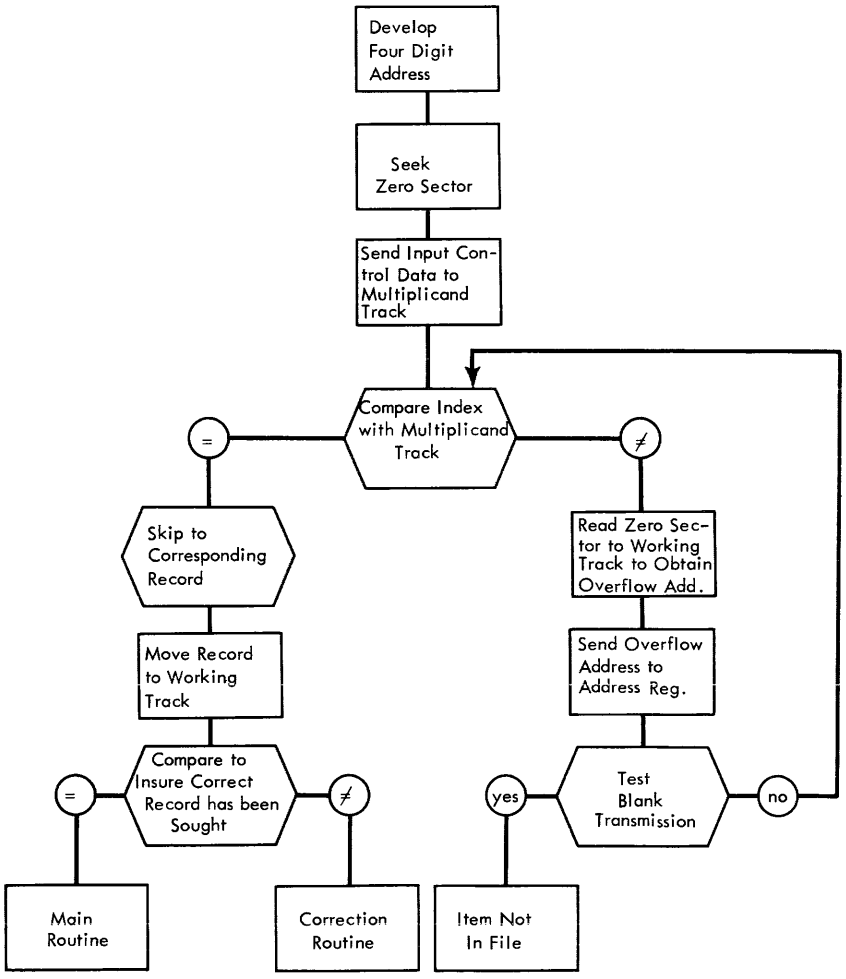


Figure 5

4. Loading the File--Prior to loading, all zero sectors in the file area should be blanked. Loading is accomplished in either one or two passes. During the first pass of two-pass loading, all records are loaded whose associated control data can be placed directly in home addresses. During the second pass, all overflows are loaded. These overflows may be placed in unused locations within the file area or in an area specifically set aside for overflows. In the latter case, loading can be accomplished as easily in one pass as in two. Locations in which to place overflows can be found by using an "address search" procedure. The loading process must also provide for placing overflow addresses, when required, in the appropriate zero sectors. As in any indirect addressing technique, items should be loaded according to expected activity, i. e., the most active items should be loaded first and the least active items last.

5. Deleting Items from the File--Deletions are made by blanking out the appropriate control data in the index record.

6. Adding Items to the File--Various techniques may be developed for handling additions. One technique is to locate the home address and:

    a. If the item can be placed in the home address, place the control data in the zero sector and the record in the corresponding sector of the track.

    b. If the item cannot be placed in the home address, proceed to the last overflow sector and place the item in an unused location (i. e., the control data in the zero sector, and the record in the corresponding sector of the track). If a new zero sector is involved, place its address in the previous zero sector.

Chaining

1. Address Capacity--Usually, but not necessarily, one item may be located in each address.

2. Nature of Items in Each Address--Each item is an actual record.

3. Retrieval of Overflow Items--Overflow items are retrieved from disk storage by means of a specific overflow address procedure. Each record includes an overflow address section, in which the location of the item is stored, if one exists, in the chain. The overflow address must be placed in each appropriate record whenever an overflow occurs during loading. This usually consists of five digits (resulting in a 95-character record) but it may be three or four digits if the chain is contained on only a few disks.

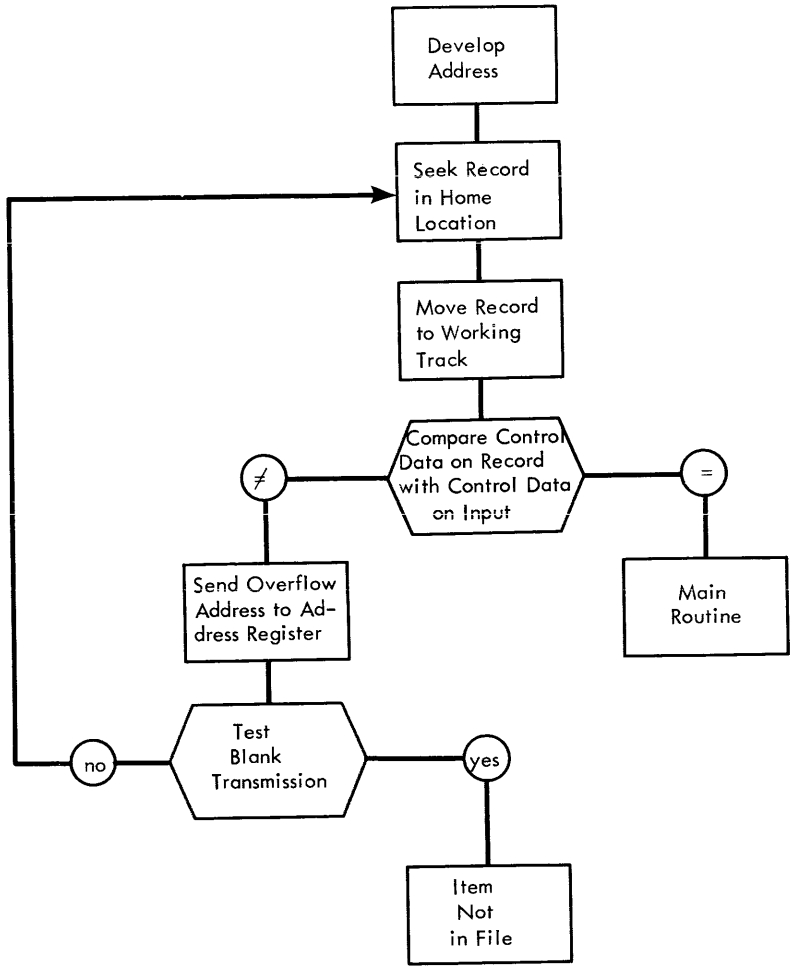


Figure 6

Space must be reserved in all records of the file for storage of these overflow addresses. If several records are contained in one sector, each record must contain space for the overflow address. Thus, when the desired item is not found in the home address, each location in the chain is examined sequentially until the item is found, or it is determined that the item is not in the file. Figure 6 illustrates a chaining retrieval process.

4. Loading the File--The disk file area to be used should be blanked before loading. The file usually is loaded in two passes. During the first pass, all items that can be placed in home addresses are loaded. During the second pass, all overflow items are loaded. These overflows are usually placed in unused locations within the file area, but can be made to an area specifically set aside for this purpose. Unused locations can be found by using an address search procedure that may be similar to the one

described under the next available address method. The location of each overflow is recorded during loading in the preceding "link of the chain" for subsequent use in the retrieval of these overflow items. As in any indirect addressing technique, items should be loaded according to expected activity; i. e , the most active items should be loaded first and the least active items last.

5. Deleting Items from the File--Various techniques may be developed for handling deletions, according to the nature of the application. One technique is to blank out all of the record except the overflow address positions.

6. Adding Items to the File--Various techniques also may be developed for handling additions. One technique is to locate the home address and:

   a. if the home address is empty, place the addition in it;

   b. if the home address is occupied, proceed to the end of the chain and by means of an address search procedure, find an unused location in which to place the addition to the chain. If an item has previously been deleted from the chain (this can be checked by a blank transmission instruction), the addition may be placed in the location vacated by the deletion.

OUTPUT CONSIDERATIONS

Output requirements should be developed early in the approach planning stages. A rough estimate of time requirements should be made to determine if they are within the limits of the system requirements.

Second to time requirements, format control and 370 forms-skipping control are the major output considerations. On the 323 Punch, selectors are available on an optional basis only; therefore, on the basic machine only one format is possible without changing the 323 control panel. The format arrangement on the output track must be planned accordingly. The 370 Printer is equipped with eight-line program selectors and four co-selectors. The line-program selectors have the capacity for six different levels of format. The program should be planned so that this selector capacity is sufficient. One must remember that each line of an MLP operation requires one level of the selector. Also, only line number 1 of an MLP may be controlled to set up variable conditions of format. The eight-line program selectors are individually controlled for pickup and dropout, and each has a capacity of eleven positions. Print control features (print start-stop, zero suppress, etc.) may require several

positions of line-program selection. A position is also required for each symbol printed that does not always print in the same position on all lines.

Form skipping on the 370 may be controlled by two methods. One, by a significant code on the output track to cause skipping only after a line is printed, and, two from the 305 panel via the communication channels. Skipping may be performed either before or after a printed line when controlled from the 305. However, it is very desirable to control skipping from the output track code whenever possible. This will facilitate checking out the 370 control panels with the Print Repeat feature, and makes the 370 more independent of the 305 operation. When skipping is controlled via the communication channels, conditions may exist in which the skip will not be executed if the 305 is stopped for some reason.

When planning the output, the following pointers should be kept in mind:

1. Print only the information that is actually needed.

2. Design forms, etc., to start printing in print position 1. A margin on the left takes print time.

3. Keep lines short.

4. Print constant data on the left, variable on the right. This helps to keep the lines as short as possible.

5. The "delta", signaling a print error, is printed three positions to the left of print position 1 (that is, two print position spaces between the delta and position 1).

6. Maximum width of forms is 16-1/4 inches from the center of one feed hole to the center of the opposite feed hole. Printing is restricted to the center 8-inch area of a 16-1/4 inch form. Any 8-inch wide area of a 12-inch form may be printed.

7. Skipping is at the rate of 25 spaces per second.

8. An original, plus seven carbon copies, is the practical maximum.

9. Print all miscellaneous heading data on one line; then other heading lines should be short ones.

10. If more than one MLP operation is used on one control panel, all corresponding MLP lines should be the same format except MLP line 1 (for example, all MLP number 3 lines must be identical).

11. Have the program determine, just before a Print signal, if a skip is required after the line is printed. If so, place the appropriate skip code on the output track before Print is signaled. This eliminates most requirements for skip control via communication channels.

12. Make the 370 self-supporting; avoid communication-channel control as much as possible.

13. Do not change control panels and forms frequently. Line format, paper forms, and control-panel wiring should be planned to make these changes as infrequent as possible.

ACCOUNTING CONTROL AND AUDIT TRAIL

The theory of in-line data processing, using random access memory, is just as dependent on sound control practices as is any other process. Therefore, every system should include a means of accounting control.

New data processing techniques, such as those offered by RAMAC, may require the development of new control techniques. The responsibility of providing adequate process control rests primarily with the programmer. The first step should be a thorough study of present control procedures. Secondly, the audit and control procedure should be reviewed.

The information gained from these two steps should provide the basis for designing the necessary controls for the RAMAC procedure.

Verification of the various machine processing functions will vary widely, depending upon the requirements of the particular application. The following are several approaches to the different areas of control:

1. Programmed machine checking. This may include use of a "proof factor", complementary multiplication, or reverse arithmetic. Another vital check is a programmed comparison between the file record and input item identification numbers.

2. Accounting controls may employ off-line checking using the RAMAC input and output cards. The output cards should be planned to contain all machine-developed data including opening and closing balances.

3. A more comprehensive check is attained by external punched-card control. Periodically, file balances are punched out and balanced against the previous balance and summary transaction cards for the accounting period.

The degree of control is an important factor and must be carefully established. It has a direct bearing on whether the system is either under-controlled and subject to undetected errors, or over-controlled and wasteful of time and of machine capacity.

Audit trail is the term usually applied to the progression of facts from one balance point to the next. Audit assumes that data is at hand, or readily available, to enable reconstruction of result data or a process for review.

Both accounting control and audit trail, with particular emphasis on the IBM 305 RAMAC, are dealt with in detail in a separate IBM General Information Manual, Form 320-0753. This publication is the result of a study by a large public accounting firm, and should be reviewed early in the program planning stage.

MULTIPLE ROUTINES

The number and nature of the various machine programs will vary with the application. However, certain generalizations can be made relating to the use of multiple routines.

First, there should be an over-all plan for the use and operation of different routines. For instance, most programs will have separate routines to load the file, unload the file, produce analyses reports, maintain the file, etc. A schedule should be made of all the routines that will be used and how they relate to each other. The best standard record format design for all programs is a major consideration.

Generally, it is best to store all instructions, constants, etc., for each program in the file. A lead card is then used to transfer the appropriate routine to the processing drum. The functions of the lead card, used to transfer in the new instructions, will be described more fully under Programmed Checking. When the programs are transferred to the drum, provision also should be made to clear the accumulators, clear all process tracks, drop out all selectors and accumulator overflow, and set up the emitter track, if one is used. This process is known as initialization.

If any one program routine should exceed the 200-instruction capacity of the program area of the drum, the additional steps may be stored in the file or on unused data storage tracks. Considerable foresight is required, when planning this type of operation, for control-panel capacity and the design of adequate controls for the operation. Usually, the first several tracks of instructions should be made permanent to the main routine. These instructions will be assigned control-panel exits that are not expanded. Also, these instructions should have firm controls to assure that other instructions are in the correct program area of the drum at all times. If the total instructions require program exits in excess of 47, selectors can be used to expand their use (see Special

Features, page 89). Generally, it is best to use the expanded exits as a group and confine their use to subroutines. When this is done, the same controls that transfer the instructions onto the program area also control the exit expansion. If the program can be designed so that the extra instructions do not have control-panel signals, it will simplify the use of the subroutines.

When multiple routines are used, there usually is a separate set of control panels and input cards for each routine. Therefore, it is necessary to positively ascertain that the programs, cards, and control panels all match. One way to do this is to assign a discrete alphabetic code to each type of input card. The card column used for the code must be different for each routine. The character selector on each control panel is wired from only those card codes that are proper to that panel. The card column used for the alphabetic code on one routine is used only for unsigned numerical values on any of the remaining routines. Thus, there is a three-way check built into the operation.

If the wrong control panel is installed in the machine, it will stop on a test-for-card code, because only the correct alphabetic codes are wired for selection purposes. If the wrong programs are used, one of the alternate card columns will be transferred to the character selector. Because this is a numerical value and only alphabetic codes are wired, the machine will stop. The same condition will exist if incorrect input cards are used. The control exit, used to test the character selector, may be wired through a combination of communication channels to test that all control panels match. If all match, a circuit is established to test the character selector; otherwise, the machine stops.

If subsequent tests of the same card code are required along the program, the numerical portion of the alphabetic character may be tested, using the numerical character selector hubs.

Another method used to interlock cards, etc., is to set up a code value for each set of programs, control panels, and cards. A card column must be reserved and used only for program routine number. For example, column 80 in the Issue cards may be coded "1", and this would be compared against a "1" contained in the Issue programs. The control panels can be checked by reserving one set of character selector hubs for this purpose and wiring only the "1" exit on the Issue panels. The 323 and 370 panels can be tied in by wiring from the "1" hub, through a combination of communication channels and back to a program re-entry point.

All programs should be planned to handle automatically as many exception routines as possible. The machine should be programmed to Stop for only those situations where operator discretion is required. In many cases, a basic error or signal routine can be planned to handle most conditions.

MACHINE PROGRAM

GENERAL

The machine program coordinates the large-capacity random access memory of the RAMAC with the input data either to update the information currently in the file, or to produce a specific type of output. An entire program can be outlined as follows:

1. Consult the input data.

2. Obtain the indicated file data.

3. Combine the input and file information to produce data for updating the file and/or data for output.

Although all programs that follow this outline can be called successful, not all of them can be termed efficient. An efficient program will perform its functions in the least amount of time. Time is one of the principal criteria for evaluating a program.

It follows, then, that a program must be constructed that will process input data (i. e., produce output data) with the greatest possible speed. While there are other considerations in a desirable program, such as the number of program exits used, drum and file space used, etc., it will generally be found advisable to sacrifice these, if possible, to achieve greater output volume.

The machine program consists of two elements: a stored program (i. e., a list of instructions preloaded in the machine) and the wired process control panel.

While the stored program is basically used to transfer data within the 305, a program must refer to the control panel for its logical decisions. Therefore, these two elements must be carefully combined to produce an efficient program.

Every decision made prior to the writing of the program, from the statement of the objectives of the installation to the details of format design, will affect the final program. In fact, early procedural proposals should be evaluated by the effect they might have on the efficiency of the machine program. It is clear, then, that an understanding of RAMAC programming is essential to proper application planning.

PROGRAMMING PRACTICE

Before considering actual programming techniques, it should be emphasized that programming is an activity in which the qualities of neatness, simplicity, and orderliness are extremely important.

A program of even moderate complexity will undergo many revisions, both major and minor, between its first draft and final version. Proper records of the instructions and track layouts at each stage should be kept. In this way, possible improvements to the program will become more apparent, accurate transferral of current results to the next version is assured, and a clear basis for discussion of the program with others is provided.

This, by no means, implies that each stage of the program should be written in complete detail. On the contrary, the actual writing of the program in machine language should be deferred as long as possible. When approaching a problem initially, it is best to begin by drawing a very general block diagram (Figure 7).

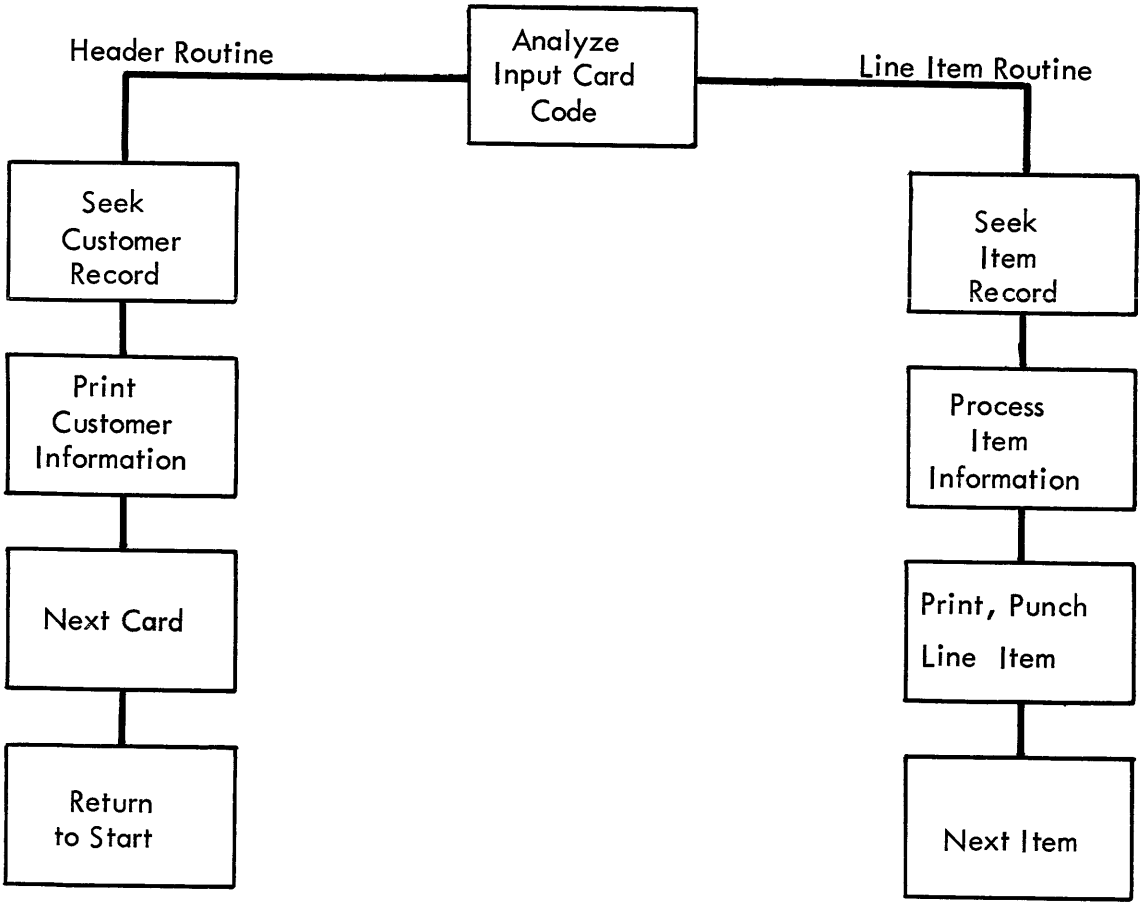


Figure 7

This practice of drawing a block diagram is already followed by many. However, at this point many programmers begin writing as shown in Figure 8.

This is too much detail for this stage of the program development. It is far better to simply write a list of program steps in very general terms. Such as:

1. Move input card to track W, feed card.

2. Move card code to character selector.

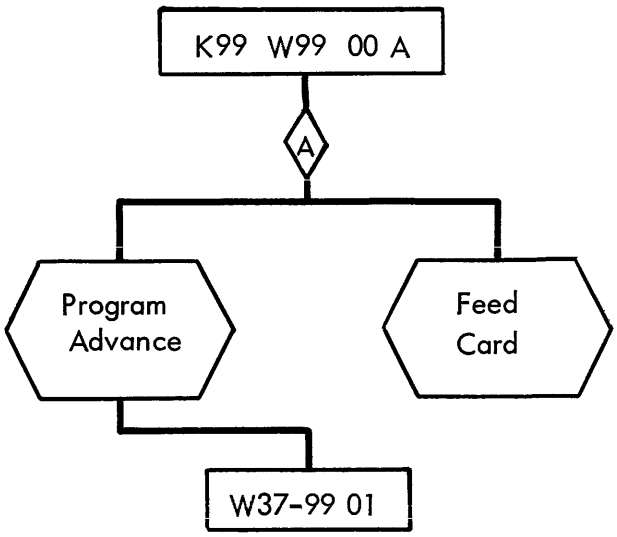


Figure 8

Writing in this way allows the programmer to keep his mind solely on the data processing problem. He need not concern himself with counting track positions, specifying number of characters, numbering program steps, etc. If the format of the input card is changed to expedite keypunching, no corresponding change is required in the program as written. If an instruction is inserted or deleted, it need not affect the rest of the program, because the steps are as yet unnumbered. In short, the program is in a form that can be revised, improved, reorganized and made more efficient.

When the program has reached what is believed to be its final form, it should be completely and properly documented. In current programming practice the most important document to be prepared is the detailed flow chart as shown on pages 74-80. This flow chart should completely illustrate the flow of each program step, including the control panel functions and their branches. It should include every detail of the program with the exception of the program steps in machine language. These would only tend to clutter the drawing. From this chart the entire program can be visualized and each step analyzed to determine its contribution to the program as a whole. After minor improvements have been made and the "loose ends" taken care of, the program step numbers should be tentatively assigned. From these step numbers the next document, the control panel function chart, can be prepared.

The control panel function chart as shown on page 81, is like a map showing the path of impulses through the control panel features. A description of the conventions used on this chart is found on pages 82 and 83. In developing this type of chart, it was originally intended to be used as an aid to debugging programs and control panels. However, it was found that many wiring errors

could be prevented and the best wiring conditions determined if this chart were made before wiring the panel.

Next, the record layouts must be carefully recorded in their final form. Examples of these are illustrated on page 84.

At this stage of the development, the program is ready to be written in machine language. The 305 RAMAC Program Instruction Sheet, Form X26-6343, should be used for this purpose. Space is provided on this form to explain each step fully, and for the wiring diagrams associated with each instruction accompanied by a program exit. The 305 RAMAC Selector Assignment Chart, Form X26-7502, should be used for notes on selector assignment. Examples of these forms are shown on pages 85, 86 and 87.

Finally, wiring diagrams can be drawn for the panels that control output format (i. e., the 370 Printer, 323 Punch, and 380 Console).

The process just described is a proven method of writing and documenting a successful RAMAC program. After gaining some experience, however, the programmer may find it convenient to place some of the burden of coding and documentation on the RAMAC itself. The RAMAC assembly program, mentioned under Programming Aids and described briefly in Section 1 of the Appendix, is a programming tool designed to relieve the programmer of much of the clerical labor associated with program preparation.

PROGRAMMING AIDS

Several programming aids have been developed by IBM. These aids are designed to assist the programmer in writing a RAMAC program in accordance with the process described. Three of these aids, which are available upon request, are:

1. RAMAC Assembly Program--A program which converts restrictive English language instructions to machine language instructions (see Appendix, Section 1).

2. RAMAC Trace Programs--Two programs that may assist in debugging the stored program and the process control panel, respectively (see Appendix, Section 2).

3. RAMAC General Purpose Board--A prewired process control panel that can be used for any data processing job on the RAMAC. While the use of this board eliminates those phases of program preparation associated with designing, wiring, and debugging the process control panel, it will frequently reduce the processing speed.

THE STORED PROGRAM

In discussing the machine program earlier in this manual, it was stated that one of the principal criteria for evaluating a program was the speed with which it is executed. In this section of the manual, five items will be discussed which, when used wisely, can become powerful tools in increasing program efficiency. It is important to study each of these areas carefully to derive the fullest possible benefit from them.

Format Design and Multiple Transfers

A basic characteristic of in-line data processing is that all information is processed in a continuous-flow type operation; that is, a single input card will affect the card reader, disk file, and output. Therefore, if the format of the input card, disk record, and output is identical, the program will be reduced to its simplest and most efficient form. However, should the arrangement of all the records be different, the program will require additional steps to rearrange and shift the data, thereby causing a longer and less efficient program.

There can be no question as to the advantages of keeping the input format and file record format identical. This is true both for loading the disks from cards or for unloading the disks to cards. Because card capacity is only 80 positions, it would be desirable to develop disk records in which 80 positions contain variable data and the 20 extra positions contain constant data. In this way, the loading and unloading of each disk record can be accomplished with only one card. If all 100 positions are variable the loading and unloading procedure will require two cards for each record and the processing time will double. If more than 100 characters must be stored, the constant data should be maintained in one sector and variable data in an adjacent sector.

In designing record formats, it must be remembered that many of the records may be consulted by more than one program.

Another important consideration in designing record formats is the numerical fields. If numerical fields in a file record can be made to line up with the accumulators, these fields can be sent to the accumulators for updating purposes with a single multiple transfer instruction. If, in addition, an input card field is to update two or more file record fields, the multiplicand track (V track) can be used to advantage. These ideas are illustrated in an example (page 88) in which six file record fields are updated by two input quantities. Note that the first two instructions would be unnecessary if four columns (instead of three) had been allocated to units on the input card.

It is often stated that output format will pose no problem, because the output information can be brought to the output track in any format whatsoever, and the required format wired at the panel with no loss in processing time. While this is true, it must be remembered that each such wired rearrangement of output data requires selection. Thus, only six 80-character output rearrangements are possible through selection at the printer.

At the punch, up to 100 positions of two-way selection are available on an optional basis only.

It would seem, therefore, that in the early stages of planning, the different types of output cards or print lines should be as identical in format as is possible with the input cards and file records. If and when these specifications are found to exceed the capacity of the standard output panel, a decision must be made as to whether additional program steps or additional control panel features will best resolve the difficulty. The area of output formats must be planned with the rest of the program, and definitely should not be considered as a final detail after the program is nearly completed.

The Magnetic Core Unit

The magnetic core unit serves as an intermediate storage for each transfer of data within the RAMAC. On one revolution of the processing drum, the desired information is read from the source track to the magnetic core. On the following revolution the information is read from the cores and recorded in the specified positions on the recording track. By considering what actually takes place during each revolution of the drum, the programmer may make more advantageous use of the magnetic core unit.

The first drum revolution is called the From cycle. The actual execution time is 10 milliseconds. As the data is read from the source track, the low-order character, as specified by the instructions, is placed in core position 99. The character to the left of this is placed in core position 98. Each succeeding character is placed, in turn, in the next core position until the specified number of characters to be transferred is sent to the core, or until the 00 position of the source track is transferred. Those positions of the core unit to which no characters have been sent retain the characters which they were storing from a previous instruction.

On the next drum revolution, called the To cycle, which is also executed in 10 milliseconds, the following occurs: Position 99 of the core unit is sent to the low-order position of the receiving track as specified by the instruction. Core position 98 is sent to the next position on the receiving track. Each succeeding core position is transferred in turn to the next position on the receiving track.

This action continues until the number of characters to be transferred, as specified by the instruction, is transferred, or until receiving track position 00 receives a character. The information that was placed in the core during the From cycle remains in the core after the execution of the To cycle.

Figure 9 shows what takes place when the instruction W04 X11 08, is given.

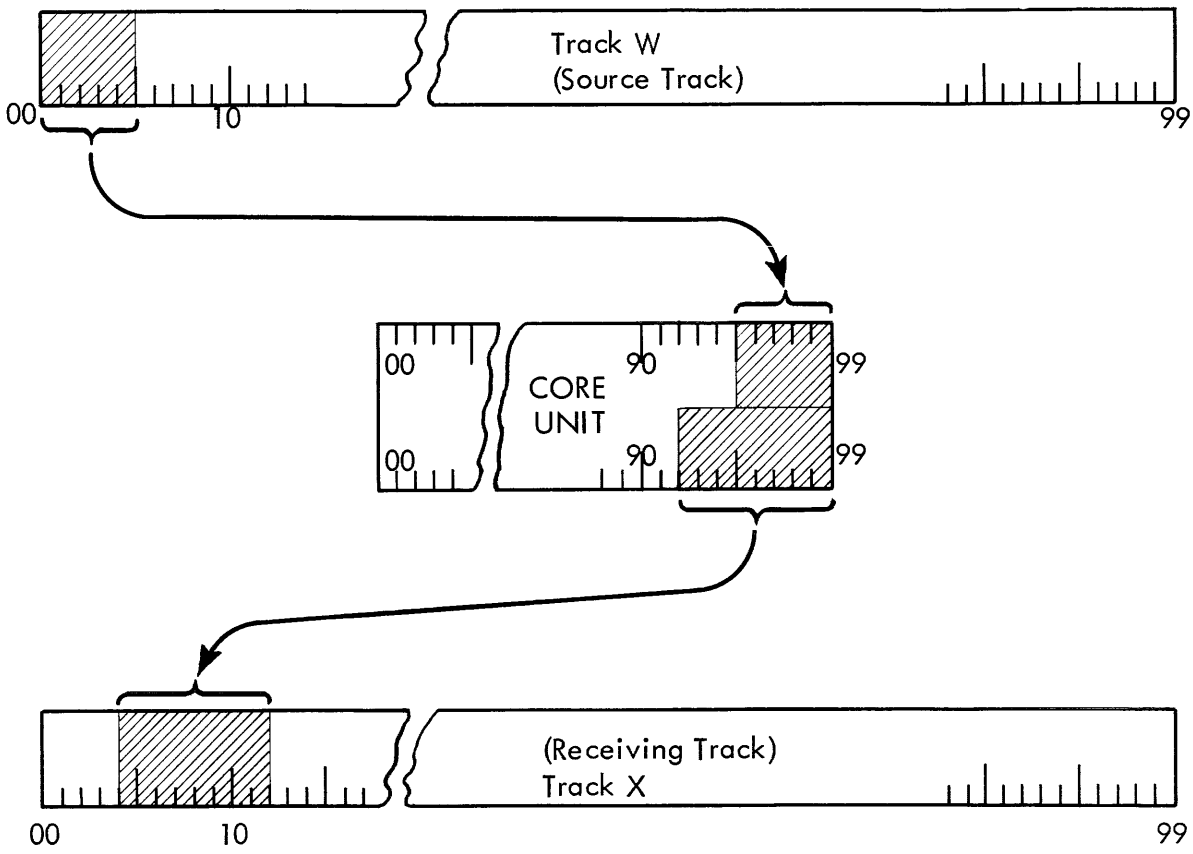


Figure 9

The effect shown in Figure 9 is that a 5-character field from track W was transferred to track X. The instruction specified eight characters to be transferred; therefore, three characters left in the core from the previous instruction were also transferred to track X.

In the example that follows, this feature is utilized and a savings of one program step is obtained. Fields A, B, and C are positioned on track W (Figure 10). It is desired to reverse their positions and place them adjacent to each other on track S (Figure 10).

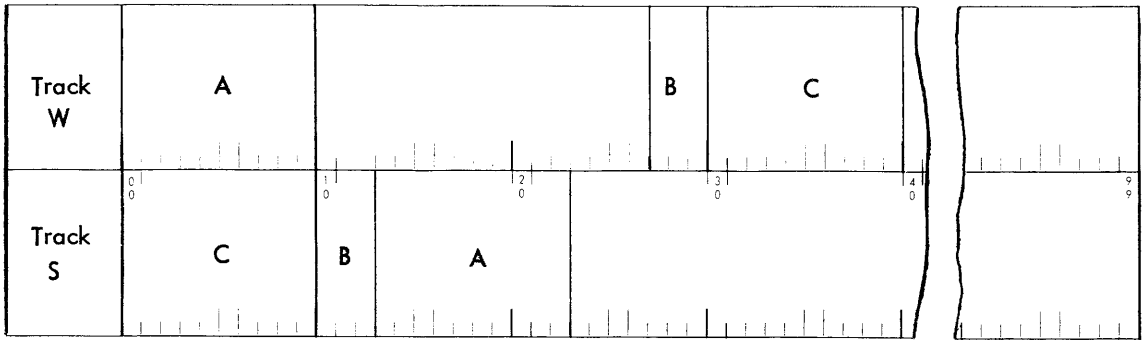


Figure 10

This is accomplished with two instructions (Figure 11).

Prog. Step	FROM			TO			No. Characters		Control	
	Track	Position		Track	Position					
xxx	W	3	9	S	0	9	1	3		
xxx	W	0	9	S	2	2	1	3		

Figure 11

The first step moves field C from positions 30-39 of track W to positions 00-09 of track S. The instruction specified 13 characters to be transferred; therefore, field B, positions 27-29, was transferred to the core unit. The second step transfers field A from positions 00-09 of track W to positions 13-22 of track S. Because the second instruction also specified 13 characters to be transferred, field B, which was left in the core by the previous instruction, is transferred to positions 10-12 of track S.

A hyphen in the From portion of an instruction can be used to transfer the data that was placed in the core by the previous instruction to any desired position. Remember that, when using the core as a "source" track, core-position 99 is always sent to the position designated in the To portion of the instruction. It is customary to write -99 as the From portion of such an instruction. Careful analysis will show when to use this feature to reduce processing time.

Two examples, in which the hyphen was used to address the core unit, follow:

Example 1 (Figure 12)

Accumulators 7 and 8 contain quantities A and B, respectively. It is desired to compute and store 2A + B in accumulator 7, and 2B in accumulator 8. Two instructions will produce this result.

Prog. Step	FROM			TO			No. Char-acters		Control		Description	Skip To Prog. Step
	Track	Position		Track	Position							
xxx	L	8	9	L	8	9	2	0			This instruction doubles the contents	
											of both accumulators. Quantity A is	
											left in core positions 80-89 and	
											quantity B is left in core positions	
											90-99.	
xxx	-	9	9	L	7	9	1	0			Quantity B, which was left in core	
											positions 90-99 by the previous	
											instruction, is added into accumulator	
											7. Accumulator 7 now contains	
											2A + B, and accumulator 8 contains	
											2B.	

Figure 12

Example 2 (Figure 13)

Each accumulator contains a positive quantity that must be multiplied by 12. Multiplication by simultaneous addition will be faster than executing 10 multiply instructions. Two instructions will again produce the desired result.

Prog. Step	FROM			TO			No. Characters		Control		Description	Skip To Prog. Step
	Track	Position		Track	Position							
xxx	L	9	9	L	9	9	0	0			This instruction doubles the contents	
											of each accumulator and leaves the	
											original quantity in the core unit.	
xxx	-	9	9	L	9	8	9	9			This instruction adds the original	
											quantity of each accumulator to itself,	
											but shifted one character to the left	
											to effect multiplication by 10.	

2A (result after first instruction) + 10A (quantity developed by second instruction) = 12A (result stored in accumulators)

Figure 13

The behavior of the core unit is slightly different when data is transferred to or from the file; that is, if R99 is used in the From portion of an instruction*, 100 characters will always transfer to the core unit, regardless of the number of characters specified by the instruction. The number of characters specified affects only the number of characters that will leave the core for the receiving track. For instance, if the instruction R99 W79 20 was executed, the entire 100-character file sector would be transferred to the core unit. Only the 20 low-order characters would continue to positions 60-79 of track W.

If R99 is used in the To portion of an instruction, 100 characters will transfer from the core unit to the disk sector, again regardless of the number of characters specified in the instruction. The number of characters specified will control how many characters enter the core prior to the transfer of 100 characters to the disk sector.

* It is customary to write "R99" in such instructions, although the machine would not behave differently for any other pair of digits.

It is because of this characteristic of the core unit that the 305 Manual of Operation states 100 characters must always be transferred to and from the disk file. However, an advantage can be obtained through this behavior. An example of this is using the core unit to assemble information. This is illustrated in the following example:

The high-order half of track A and the low-order half of track B are to be assembled and transferred to a disk sector. This can be accomplished with two instructions (Figure 14).

Prog. Step	FROM			TO			No. Char-acters		Control		Description	Skip To Prog. Step
	Track	Position		Track	Position							
xxx	A	9	9		9	9	0	0			Transfer the contents of track A to the core unit.	
xxx	B	9	9	R	9	9	5	0			Transfer the 50 low-order characters from track B to the core unit and transfer contents of core to the disk sector. The disk sector now contains the high-order half of track A and the low-order half of track B.	

Figure 14

A compare instruction, using the disk address as the From portion of the instruction, will also transfer 100 characters to the core unit. If the instruction R99 W04 05 A1 is executed, positions 95-99 of the disk sector will be compared with positions 00-04 of track W and the result stored on the compare selector. The entire disk record will be left in the core.

Another example of saving program steps by making use of the core unit is the example in which positions 00-94 of track X are to be placed in the last 95 positions of a file record. The disk address is stored in the first five positions of the record. The instructions shown in Figure 15 are used.

When the record was transferred from the disk file to track W, the disk address was retained in core positions 00-04. It remained there undisturbed during the compare instruction, and was recorded back in the file, with the record, during the execution of the last instruction.

Prog. Step	FROM			TO			No. Char-acters		Control		Description	Skip To Prog. Step
	Track	Position		Track	Position							
025	L	9	9	J	9	9	0	5			Seek record developed in accumulator 9	
026	R	9	9	W	9	9	0	0			Transfer record to track W	
027	L	9	9	W	0	4	0	5	P	1	Compare address with address	
											developed in accumulator 9.	
028	X	9	4	R	9	9	9	5			Transfer record to disk file.	

Figure 15

In using the core storage unit to retain information between instructions, there is one point that must be kept in mind: All 100 positions of the core unit are used during certain console operations. Inquiry (manual and programmed) and console-initiated transfers of information between drum and typewriter use the core buffer the same as does a RAMAC instruction. In the last example, if the RAMAC program were halted after execution of the second instruction, it could be restarted at the third instruction only if no manual inquiry or drum track read operation was initiated at the console. Inquiry would of course, require a restart at the first instruction.

The Accumulators

An accumulator feature, use of which is made frequently in an address conversion formula, is its ability to disregard the zone (X and 0) bits of all incoming alphabetic characters except in the sign control position. (The sign control position is the first, or low-order, character read into an accumulator.) If J6DV7 is reset in an accumulator, the result will be +16457.

If 222BSK were entered in the accumulator, the result would be 222222-. When a negative number is read out of an accumulator, the machine will add an X bit to the lowest order digit as the data is enroute from the accumulator track to the core unit. Thus, if 1023- were read out of the accumulator, the result would be 102L.

Programmers familiar with the zone-stripping property of the accumulators can obtain some limited use from it. For example, a program to load file records sequentially from input cards is developed. In this program an accumulator is used to produce the next disk address to be loaded and also to stop the machine at the appropriate time. This is accomplished by initially placing the number of sectors, and the address of the first disk sector to be

loaded in a reset accumulator. During each pass of the program loop, 99999 is subtracted from the accumulator. This increases the disk address by one and decreases the number of sectors to be loaded by one.

Thus, if accumulator 9 contains	—	000145	12400
and the arithmetic instruction is executed	—	-	99999
the result will be	—	000144	12401

After all the sectors have been loaded, and the accumulator becomes negative, the program stops. To eliminate storing the constant 99999, the programmer begins investigating his program instructions and finds that instruction 198 reads X99 R99 00. He immediately sees that 99R99 can be used as a constant, and he writes his arithmetic instruction as I85 M99 05.

Care must be exercised in writing instructions that address more than one accumulator if negative quantities are involved. For example, initial quantities of +5, -2, and -3 are to be stored in accumulators 0, 1, and 2, respectively. The required constants are stored on track H (Figure 16) and separated by blanks. Character K (X-2) and L (X-3) are used to designate -2 and -3.

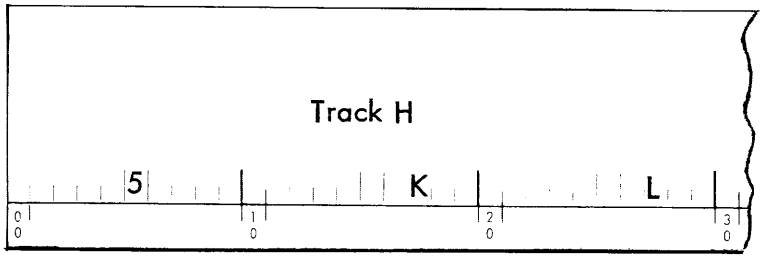


Figure 16

If the one instruction, H27 L27 23 b5 were executed, accumulator 1 would receive a +2 rather than the desired -2. The reason is the lowest order digit read into accumulator 1 was a blank, which is not accompanied by an X-bit; therefore, accumulator 1 is positive. To rectify this, a hyphen (X-bit) may be placed in position 19. Another solution is to set the constants as shown in Figure 17 and use the instruction H27 M27 23 b5.

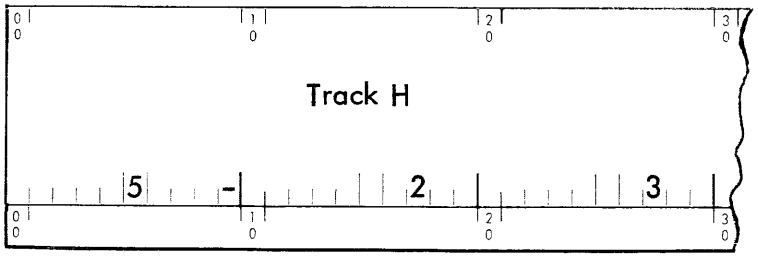


Figure 17

The RAMAC, maintaining algebraic sign control, will enter these digits as +5, -2, -3.

Another example of the care to be taken in addressing several accumulators with one instruction is the following. A programmer decides to effect a multiplication by eleven by using the instruction L99 L98 49. In this case, five accumulators are affected. To produce the proper result, each accumulator affected must be positive with the exception of the last (or right hand) one; this may be negative. The reason is that when the digits are read back from the core unit into the accumulators, they are shifted one position to the left. This means that on each negative accumulator the X-bit will be ignored, because it will not be replaced in the sign control position.

Prog. Step	FROM			TO			No. Characters		Control	
	Track	Position		Track	Position					
xxx	L	9	9	L	9	9	5	0		
xxx	-	9	9	L	9	9	5	0		

Figure 18

Multiplication by three, however, can be obtained on five accumulators, regardless of the sign of each individual accumulator. This method uses the advantages of the core unit described in the previous section. The instructions are shown in Figure 18.

Proper results are obtained in this method because an X-bit, in the sign control position is always returned to the same position, maintaining the proper sign.

When reading out of an accumulator, it is best to avoid the read out and reset code (M). A basic rule in programming is to retain any stored data until it is necessary to use the storage space for more data. This will facilitate checking the cause of an unscheduled machine stop or error. Also, the familiar programming error (forgetting to reset an accumulator before storing data there) is less likely. The accumulators can generally be reset with the same instruction that brings data in to them.

The Instruction Register

Each instruction in the RAMAC is executed from the instruction register to which it is sent just prior to its execution. The instruction register is a 10-character storage unit in which each position is assigned to one of the instruction positions. This component is able to strip the zone bits in its second, third, fifth, sixth,

seventh, eighth, and tenth positions. The remaining positions, the first (From track), the fourth (To track), and the ninth (program exit) positions will store alphabetic characters. Thus, if the characters BILLSbPAGE were written as an instruction, they would arrive at the instruction register as B93 L20 71 G5 and would be executed as such. (Each character in the instruction register can be determined, as the instruction is executed, by the use of the bank of lights labeled Instruction on the 380 Console.) This property of the instruction register is perhaps difficult to apply to advantage. Nevertheless, the following example is taken from an actual RAMAC program.

A subroutine is to be executed exactly three times for each pass through the main routine. The programmer has no accumulator available to count three, nor does he want to use any other logical device on the process control panel in this application. The last instruction in the subroutine adds ten characters from positions 00-09 of track Z to accumulator 0.

Figure 19 represents the chosen solution.

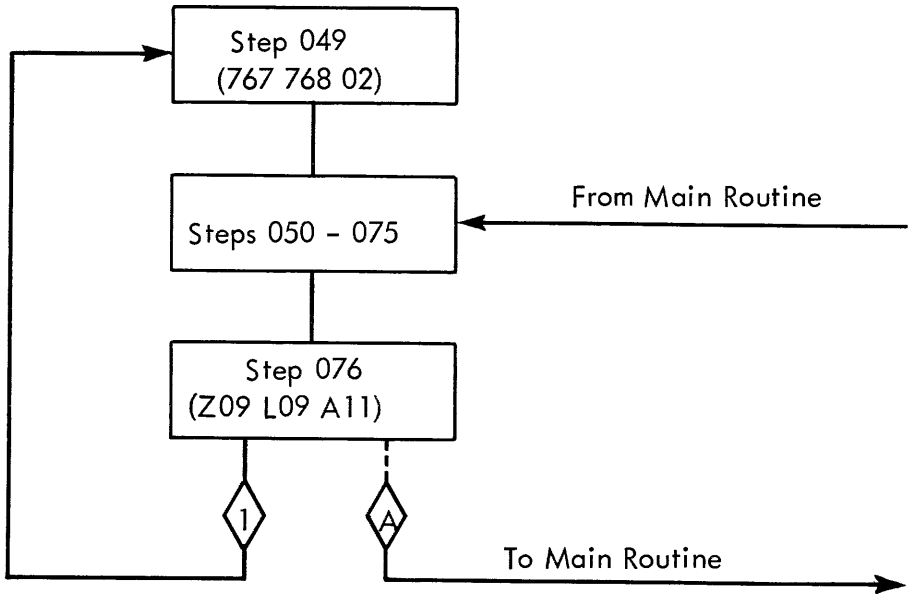


Figure 19

The first time through the routine, step 076 is executed as Z09 L09 11 1. Program Exit 1 transfers the routine to step 049 which modifies instruction 076 to read Z09 L09 AA 1. When executed the second time, the action is the same--ten digits are added to the accumulator and the routine transfers to step 049. At the end of the third pass through the routine, the instruction reads Z09 L09 AA A. This time, Program Exit A, which transfers the program to the main routine, is used.

Instruction Modification

A programming facility, frequently overlooked by RAMAC programmers is that of instruction modification; that is, the ability of a stored program computer to change its own instructions during the course of data processing. This powerful feature was illustrated in the last example given in the previous section.

There are two types of instruction modification:

1. The specific modifications are known in advance to the programmer. They occur identically each time the routine is executed.

2. These modifications cannot be predicted in advance because they depend on the data being processed at the time the modifications occur.

Instruction modification of the first type is illustrated in the following example. Twenty-five four-digit quantities are stored sequentially on track Z. It is required to accumulate their sum in accumulator 0. Using instruction modification, this can be performed with four program steps and two program exits.

Prog. Step	FROM			TO			No. Characters		Control	
	Track	Position		Track	Position					
110	W	0	2	L	1	1	0	3	P	5

Figure 20

The solution is shown in Figures 20, 21, and 22.

W00-02 contains bb3. This step serves three functions: (a) it resets accumulators 0 and 1; (b) it sets the initial constant of 03 in accumulator positions 10 and 11; (c) Program Exit P drops out the accumulator overflow.

Prog. Step	FROM			TO			No. Characters		Control	
	Track	Position		Track	Position					
111	L	1	1	A	2	2	0	2		

Figure 21

Step 111, shown in Figure 21, inserts into step 112, the location on track Z of the first of the 25 quantities to be added into the accumulator. After each quantity is added into the accumulator, this number is increased by four to produce the location of the next quantity to be loaded (step 113).

Step 112 adds the desired quantity to accumulator 0. W03 contains a 4. Step 113 adds the 4 to accumulator 1, which will produce the location of the next quantity to be loaded into accumulator 0. Program exit Q is used to test the accumulator overflow. If no, the subroutine transfers back to step 111. If yes, the subroutine returns to the main program routine (Figure 22).

Prog. Step	FROM			TO			No. Characters		Control	
	Track	Position		Track	Position					
112	Z			L	0	9	0	4		
113	W	0	3	L	1	1	0	1	Q	

Figure 22

Step 110 sets up the initial conditions necessary to the routine. Such a step, or set of steps, is called the initialization of the associated routine, and is a necessary element in all loop-type routines. Programs that involve instruction modification sometimes require initialization of the instructions to be modified, although this is not necessary in the present case.

Had it been positively ascertained before time that none of the data fields of the preceding example were zero or blank, a track slide operation could have solved the problem. The sliding method is sometimes more desirable; although in terms of program speed and space required, it is about equal to the instruction modification method.

The preceding example is an illustration of the first type of instruction modification. Note that the desired result could have been achieved by using 25 steps to send each quantity to the accumulator. This solution is much faster, but requires 25 steps as opposed to 4 steps and 2 program exits. It is characteristic of routines using the first type of instruction modification that identical results can be achieved, usually with greater speed, without instruction modification if one is willing to store the additional instructions, which are known in advance by the programmer.

The situation is different in the case of modification routines of the second type. Here the use of instruction modification reflects the requirement that the instructions to be performed depend upon the data being processed. If modification is not used, some other method of satisfying this requirement must be found.

Invariably the alternative method involves the use of additional logical facilities on the process control panel. The reason is evident when one reflects that the control panel features provide a means for instruction modification of a limited sort. Consider, for instance, a program exit wired to the In hub of an accumulator sign selector. The +, 0, and - hubs of the selector are wired to different

functions. The sign of the number in the accumulator, which is a part of the data being processed, determines which of these functions will be executed by the program exit. Thus, the instruction carrying the program exit may be modified by the data.

A common problem in RAMAC programming is that the programmer finds his process panel facilities (program exits, distributors, character selectors, etc.) used up when the program is far from complete. This difficulty can frequently be avoided by judicious use of instruction modification of the second type. Two examples of this follow.

Example 1

Part of one disk contains 10-character records stored ten to a sector. To address each record, a 6-digit address is required. The first five digits denote the disk address and the last digit indicates the proper 10-character field within the sector. The program requires the correct 10-digit field to be selected and placed in positions 00-09 of track X. The address of the 10-digit field is in positions 00-05 of track W.

In using the character selector to perform this operation the disk address is sent to the address register. Next, the sixth digit of the address is stored in the character selector, and the character selector In hub is impulsed. Each of the 10 digits on the character selector is wired to the program step, which will transfer the 10-digit record, corresponding to the digit stored, to positions 00-09 of track X.

Prog. Step	FROM			TO			No. Characters		Control	
	Track	Position		Track	Position					
010	W	0	4	J	9	9	0	5		
011	W	0	5	1	3	1	0	1		
012	R	9	9	Y	9	9	0	0		
013	Y		9	X	0	9	1	0		

Figure 23

Using the instruction modification method to perform this operation, the instructions shown in Figure 23 are used.

Instruction 011, places the 10-digit field designation in step 013, which transfers the desired data to positions 00-09 of track X.

In addition to being faster, the instruction modification method is far more economical in terms of drum space, and leaves the character selector free for some better suited purpose.

In the preceding example, the modifying data (sixth digit of the address) was used to modify the appropriate instruction directly. Frequently, however, this data must be adjusted slightly before modification. An "emitter" track may be useful in solving otherwise perplexing programming problems.

Example 2

In Example 2, a billing application, a single-digit code, placed in position 99 of each item record, is used to designate the discount to be applied to the item extension. The codes for the discount are as follows:

1 = 3%, 2 = 5.5%, 3 = 1.5%, 4 = 6%, 5 = 4%

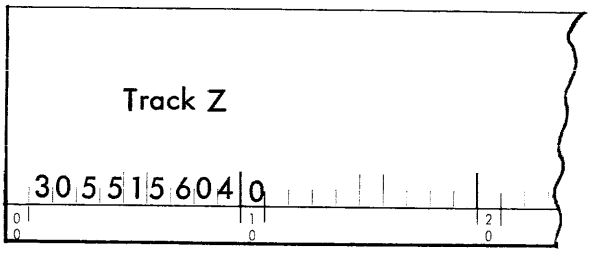


Figure 24

These discounts are stored in order in positions 01-10 of track Z. in Figure 24. This is known as an emitter track.

After the item record is sought and transferred to track X, the item extension is computed and sent to the multiplicand track. The four instructions shown in Figure 25 are used to send the proper discount percentage to the multiplier.

Prog. Step	FROM			TO			No. Characters		Control		Description	Skip To Prog. Step
	Track	Position		Track	Position							
020	X	9	9	L	9	9	0	1		5	Reset-Add item code to accumulator 9.	
021	L	9	9	L	9	9	0	1			Double contents of accumulator 9.	
022	L	9	9	2	3	2	0	2			Modify instruction 023 to select proper discount	
023	Z			N	9	9	0	2			Multiply extension by discount.	

Figure 25

Here, again, the character selector could have been used. It would have resulted in a slightly faster routine (100 ms as against 120 ms), but would have required six instructions, two program exits, two double distributors, and a character selector. The programmer must decide in such cases whether the 20 milliseconds saved is worth the price.

Familiarity with instruction modification possibilities is valuable even in the early stages of application planning. Some decisions made at this time are difficult to change later on, because of printing schedules, etc. The alert

programmer can frequently suggest product codes, for instance, that will result in time savings in the RAMAC program.

An example of this is the following billing application in which each item belongs to one of eighteen product classes. For each item ordered, two records must be updated: the item record and its associated 50-character product class record. Because of the very high output volume requirements, it is decided to batch the input orders. Before the processing begins, a header card will transfer the product class records to drum tracks A through I. (It is estimated that less than 100 instructions will be required on the drum.) The updated records will be returned to the file when processing is complete.

This scheme will speed up the process by requiring only one seek instruction per input item instead of two. It remains to assign the eighteen product class codes, one of which will be a part of each item record. For this purpose, it appears that the 18 pairs of digits, A4, A9, B4, B9...I4, I9 will constitute a code assignment which results in efficient programming. If the appropriate product class code is stored in positions 98 and 99 of each item record, the program might be as follows:

The item record is sought, sent to track X for processing and returned to the disk file. Track X retains all the item record data allowing the next step to read as shown in Figure 26.

Prog. Step	FROM			TO			No. Char-acters		Control		Description	Skip To Prog. Step	Time in milli-seconds
	Track	Position		Track	Position								
060	X	9	9	6	2	1	0	2			The product class code is inserted in step 062		
061	X	9	9	7	8	4	0	2			The product class code is inserted in step 078		
062			9	Z	4	9	5	0			The product class record is sent to a working track		

Figure 26

Steps 063-077 process the product class record, and step 078, Z49--9 50, returns the product class record to the drum.

In this example a careful choice of product class code enables steps 060 and 061 to modify steps 062 and 078 directly with the code itself.

THE PROCESS CONTROL PANEL

An exceptionally efficient machine program will depend upon the proper use of both the stored program and process control panel. The proper use of a program step can save process control panel space and operating time, while a carefully planned and wired process control panel can save program steps and also operating time. However, the machine program will only be as efficient as the weaker of the two elements. Both must be skillfully combined to produce an efficient machine program.

To make better use of the process control panel it is best to begin by understanding its functions fully. The process control panel is used to return to the stored program once a program step has referred to the panel for a logical decision. It is used to skip from one stored program step to another, to initiate the input or output cycle, to start or halt processing, or to store information on latch-type selectors. A means of communication is provided between process control panel and the print and punch panels. Special uses of the access arm can be obtained through the control panel. It will also control, in a limited way, the mode of operation of the 305.

The logical devices that are available on the process control panel monitor the data being processed so that the control panel functions just mentioned may be selected by the nature of the data.

In learning to use the process control panel properly, programmers are frequently confused by the apparent variety of hubs available. Actually, each hub belongs to one of six categories. In the following listing of these categories, two or more hubs internally connected together (o—o or o—o—o) will be referred to as one hub.

1. Exit

   An exit hub will emit an impulse 10 milliseconds in duration under specified conditions. For example, Program Exit A emits an impulse following the To cycle of a stored program instruction containing the character A in its program exit position. Cycle Delay 7-Out emits an impulse 30 milliseconds after its In hub has received an impulse.

2. Entry (or Function)

   These hubs will accept an impulse that will either return the program to a stored program step or initiate a specific function. For example, Punch, when impulsed, causes the machine to punch an output card. Record Advance In causes the access mechanism to advance to the next sector on the same disk tracks (4 to 5, 2 to 3, etc.). An impulse is emitted from Record Advance Out (an exit hub) 30 milliseconds later. Other function

hubs include Print, Stop, Feed Card, Type, Reset Stop, Program Entry Hundreds, Tens, Units.

3. Logical Device

These devices include Accumulator Sign and Overflow, Blank Transmission, Compare Selector, and Selectors. Each device is comprised of two or more identical "sections." A section is a set of hubs, one of which is called the In (or Common) hub. The remaining hubs (two or more) are called "status" hubs, and have various designations, depending upon the logical device to which the section belongs. The separate sections of a logical device are electrically isolated from one another. The In (or Common) hub of each section is at all times connected to one, and only one, of its associated status hubs. The status hub to which the In (or Common) hub is connected is determined by the status of the element that controls this particular logical device. This "control element" generally has a name similar to the device it controls. For example, accumulator 1 is the control element for the accumulator 1 sign selector. When accumulator 1 is negative, each negative status hub (-) of the accumulator 1 sign selector is automatically connected internally to its associated In hub. No two status hubs of a logical device are ever connected together internally.* The compare selector is a logical device having ten sections. Its control element is the compare selector which is positioned by a RAMAC instruction having a 1, or 3 in its last (tenth) position, as explained in the 305 Reference Manual. The In hub of each section is always connected to one of its two status hubs, = or $\neq$. Selector 1 is a logical device having two sections. The control element, selector 1, is positioned by the arrival of an impulse at either its PU or its DO hub. The common hub of each section is internally connected to the Normal (N) hub if the DO hub was last impulsed, or to its Transfer (T) hub if the PU hub was last impulsed.

4. Distributor

All distributors contain In and Out hubs. A single distributor has one In and one Out hub and a double distributor has one In hub and two Out hubs. An impulse received at the In hub of a distributor appears simultaneously at the Out hub(s). An impulse arriving at an Out hub, from a source outside the distributor, does <u>not</u> appear at the In hub of the distributor. An impulse arriving at the Out hub of a double distributor will <u>not</u> appear at its In hub or at the other Out hub.

* The character selector is a special logical device that does not conform to the characteristics claimed above. It will be discussed separately.

5. Mode Switch

A mode switch is a pair of hubs that control the mode of a specified RAMAC operation. For example, the File hubs, when connected, permit data to be written in the disk file. When not connected, any attempt to write in the disk file will produce a stop on a file interlock. Other mode switches include ALC, ITI, and Inquire On.

6. Communication

These hubs provide a means of communication between the process control panel and the print, punch, or typewriter panels. An impulse delivered to a communication hub on the process control panel is transmitted to the corresponding hub on the other control panel. For example, Punch Communication 2, when impulsed, transmits this impulse to Punch Communication 2 on the 323 Punch panel. Similarly, an impulse delivered to the latter hub will appear simultaneously on the process control panel.

A full description of each of the control panel hubs is contained in the 305 Reference Manual.

The following paragraphs discuss some of the collective properties of the hubs, pointing out rules of operation frequently ignored, as well as useful properties frequently overlooked.

The symbols and legends of each of the following diagrams are self-explanatory with the exception of the convention used for distributors and program entries. They are shown in Figure 27.

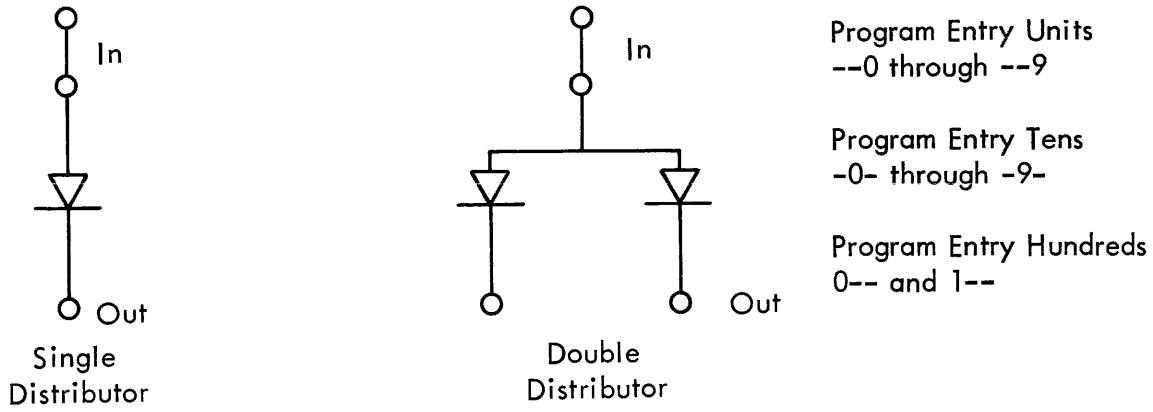


Figure 27

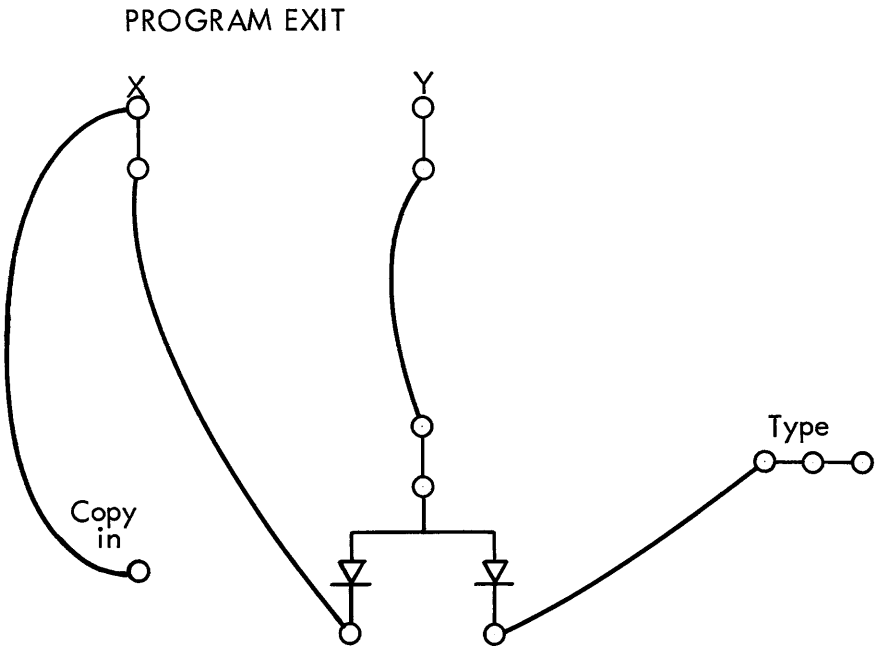


Figure 28

Two or more function hubs may be impulsed simultaneously; however, any impulse, wired to more than one type of function must be distributed to all branches. Distributors allow an impulse to travel in one direction only, thereby eliminating the possibility of back circuits or of an undesired function being impulsed. In Figure 28, the distributor prevents Program Exit X from impulsing Type.

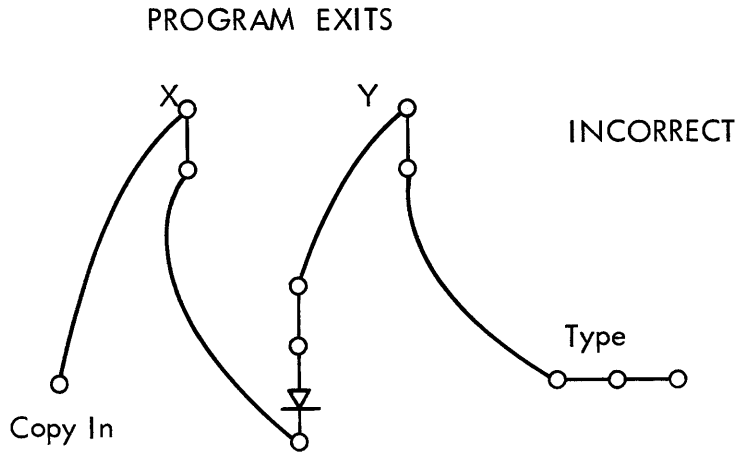


Figure 29

No attempt should be made to deviate from this rule. For example, it may appear that the same result can be obtained if Program Exits X and Y are wired through a single distributor as shown in Figure 29. However, even if Type is never impulsed from another source, such a connection can produce electrical circuit difficulties resulting in improper operation.

In one instance, an exception may be made. Two or more selector PU (or DO) hubs may be connected together and treated as "one type of function." Thus, the wiring in Figure 30 is correct.

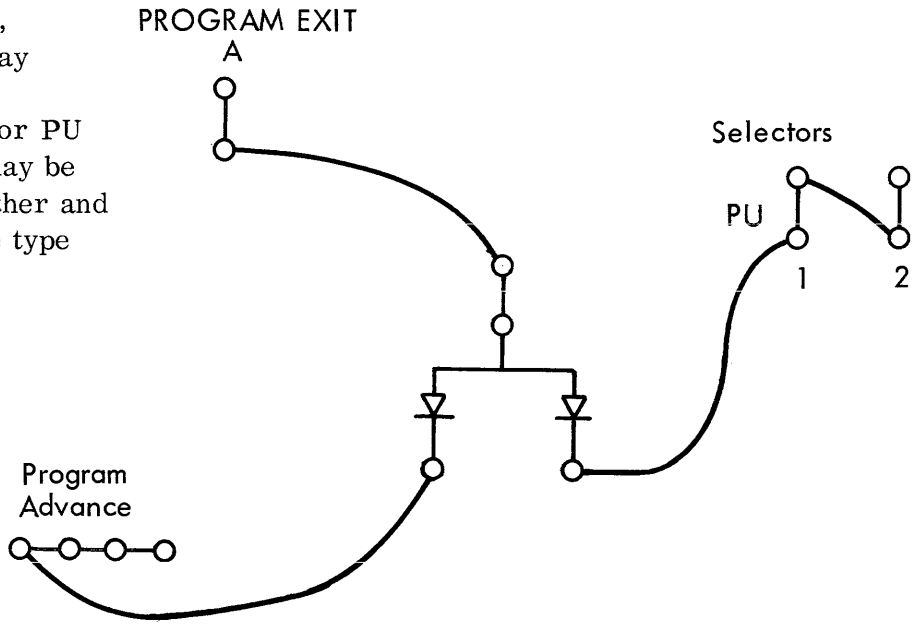


Figure 30

If, however, it is later found that another program exit must pick up selector 2 only, the wiring must be revised as in Figure 31.

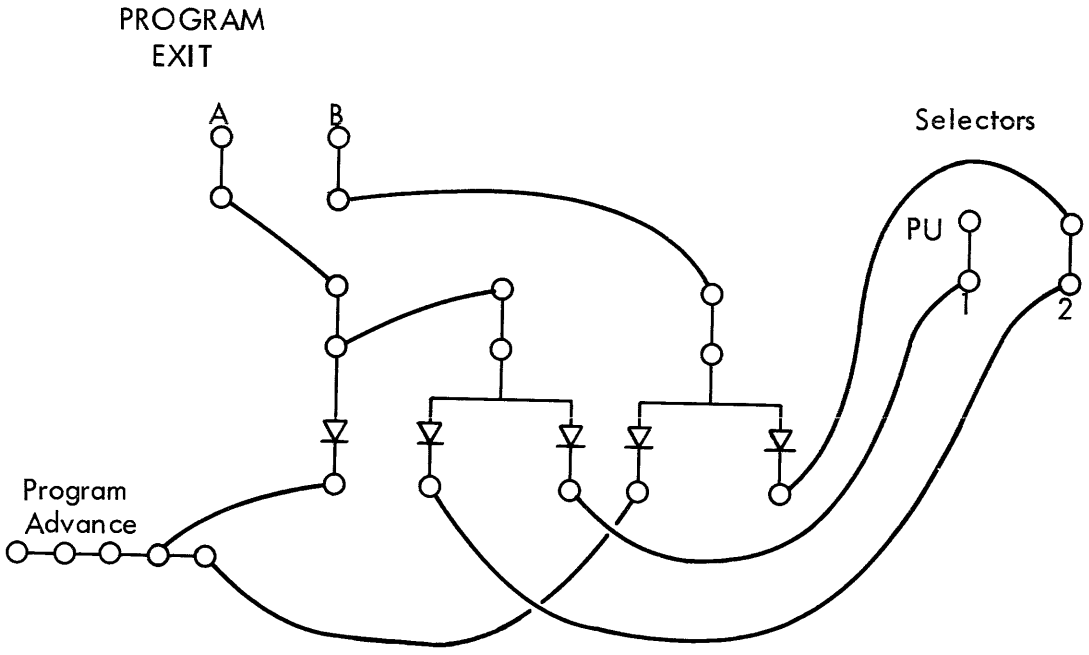


Figure 31

When impulsing two functions simultaneously via a distributor, care must be taken to avoid wiring improper combinations. For example, it is clearly incorrect to impulse Program Advance together with Program Entry Tens 2 and Units 7. It is also wrong to attempt to position a logical device while at the same time attempting to send an impulse through the status hubs. The wiring of Figure 32 is incorrect because the status of the accumulator overflow is being changed by an impulse passing through one of its status hubs. This error can be corrected by wiring through a cycle delay before wiring to Drop Out and Program Advance.

To obtain a program transfer to some desired step, Program Entry Tens and Units must be impulsed simultaneously. Failure to do so will produce a machine stop. The same result will also be obtained if Tens only or Units only is impulsed.

When Stop is impulsed simultaneously with other functions, these other functions will be executed prior to the stop. Thus, if on program step 028, Stop is impulsed together with Program Entry Tens 4 and Units 5, the machine will send instruction 045 to the instruction register and then stop. The console lights will indicate a stop on 045.

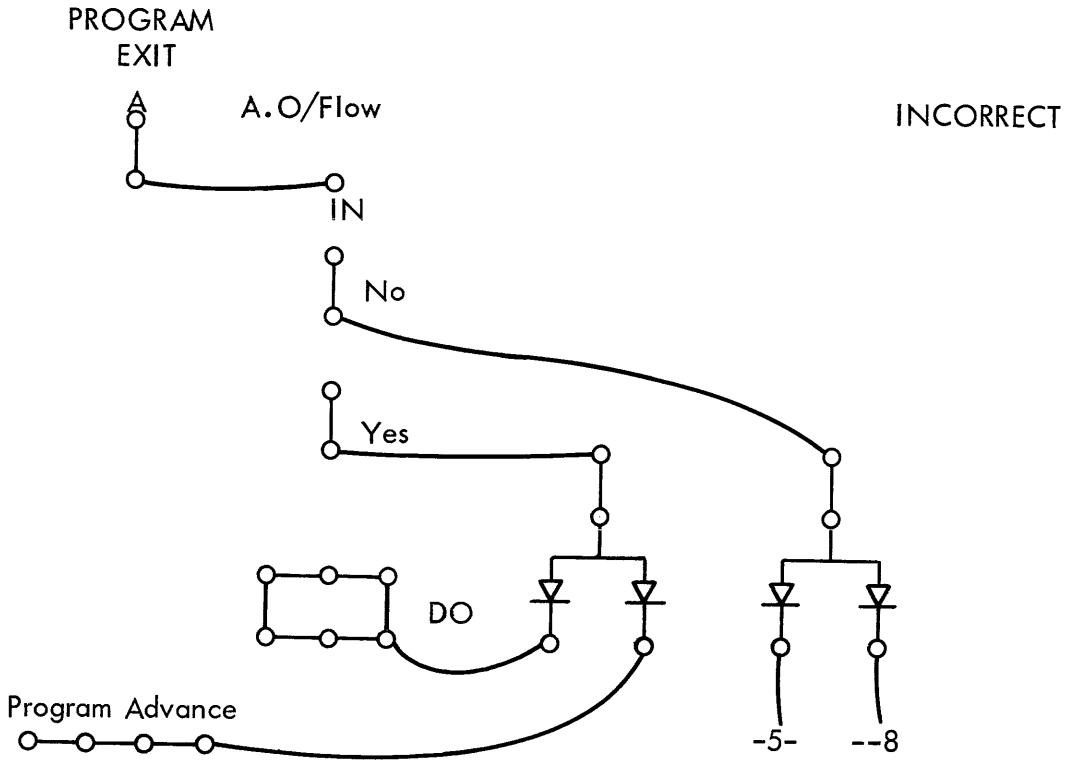


Figure 32

The remaining important rule concerning distributors is that the Out hub of one distributor must not be connected to the In hub of another. An equivalent statement is that an Exit hub must not be connected to a Function hub via two distributors in tandem. Figure 33 is incorrect.

To obtain the results attempted in Figure 33, Program Exit A can be wired through two double distributors to Print, Program Entry Hundreds 1, Tens 6, and Units 5. Program Exit B will require two double distributors and a single distributor to impulse Print, Type, Program Entry Hundreds 1, Tens 6, and Units 5.

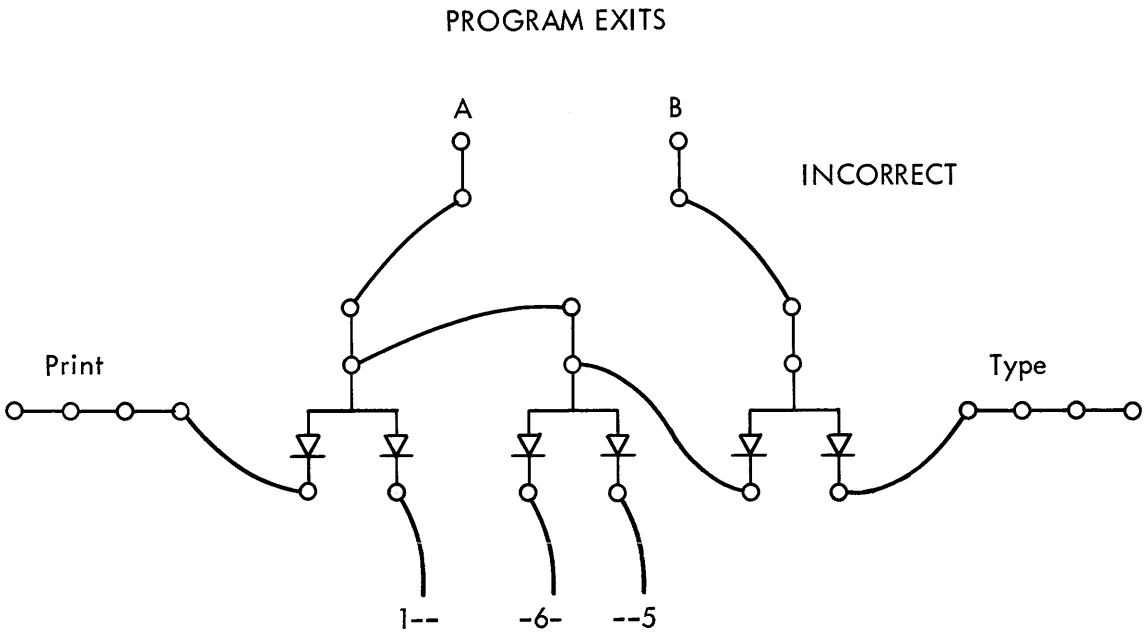


Figure 33

An alternate method that is useful when distributors must be conserved is shown in Figure 34.

Notice in Figure 34 that a distributor was not required when impulsing Cycle Delay In. This is possible because the Cycle Delay In hubs are distributed internally. However, if two impulses are split-wired into one Cycle Delay In hub, and there is a possibility of a back circuit; either distributors should be used, or the impulses should be wired to separate Cycle Delay In hubs.

There is no objection to the impulse of one Exit hub arriving at another Exit hub (or hubs), whether filtered or unfiltered. This is illustrated in Figures 27 and 34.

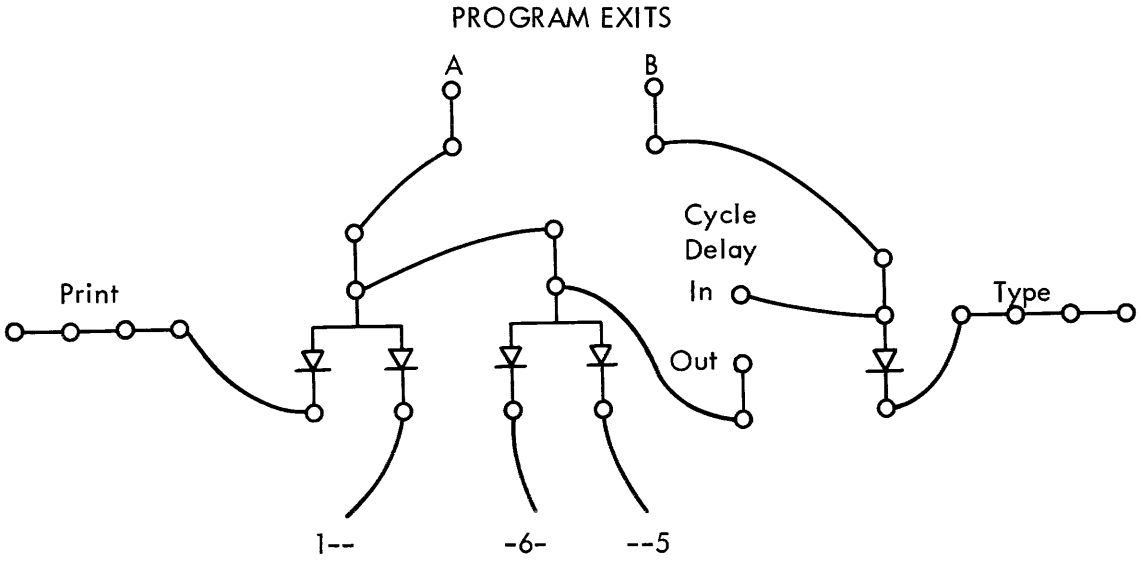


Figure 34

The character selector is a very powerful logical device, in that many more status hubs are available in each of its four sections than in any other logical device section. The four sections are not, however, identical. The larger, or alphamerical section, has one In hub and 48 independent status hubs. Each status hub corresponds to one of the alphabetic, numerical, or

special characters used in the RAMAC. The three smaller sections each have one In hub and 13 status hubs. The status hubs in the three smaller sections differ slightly from other logical elements, in that one or two status hubs may be connected internally with the In hub. This means that, on alphabetic characters, the In hub will be connected internally with a zone hub and a digit hub. For example, if the character "A" is stored in the character selector, three hubs (In, 12, and 1) are connected internally. When two status hubs are connected, one will always be a zone hub and the other a digit hub. No two-digit status hubs, (1-9) can ever be connected together, nor can two zone status hubs (12, 11, or 0) ever be connected together. Blank (BL) can only be connected to the In hub.

Special characters, whose IBM card code numerical portion is 3-8 or 4-8, will connect the zone designation and the upper punch only. For example, the character % (0, 4, 8) will connect the In hub with zone 0 and digit 4. Care must be exercised when storing special characters in the character selector. Consider the following example. A program is developed in which Program Exit A is used to transfer to step 100, 110, or 120, depending on whether the code in the character selector is 0, 1, or 2. The other sections of the character selector perform other functions. Program Exit Z is used in step 098 to transfer directly to step 010. The panel is wired as shown in Figure 35.

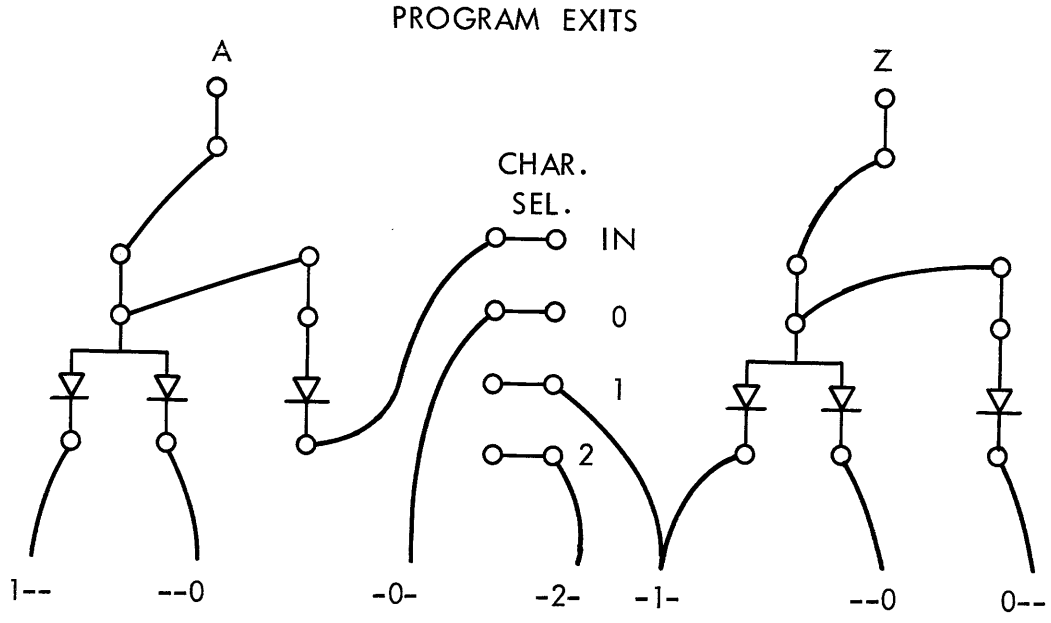


Figure 35

After the program has performed successfully for several days, a machine stop suddenly occurs on step 098, because for the first time a slash (/) has been stored in the character selector, internally connecting status hubs 0 and 1. When Program Exit Z emits an impulse at step 098, Program Entry Tens 0 and Tens 1 become impulsed simultaneously.

A more frequent problem in utilizing the character selector is that of using a card code to select a program routine. In a typical application, card codes 1, 2, 3, 4, and 5 are used to transfer to five different steps. All other card codes are used to transfer to a "wrong card code" routine at step 134. The character selector is wired as in Figure 36.

This method is effective only if a numerical card code is used exclusively. If an alphabetic character should be used in the card code, the machine will attempt to transfer to two different program steps simultaneously and a machine stop will occur.

In those applications where numerical characters are not used exclusively in the card code, the alphamerical section of the character selector can be utilized or the cycle delay device can be used as a logical element (see 305 RAMAC Bulletin 52, Form 328-0659).

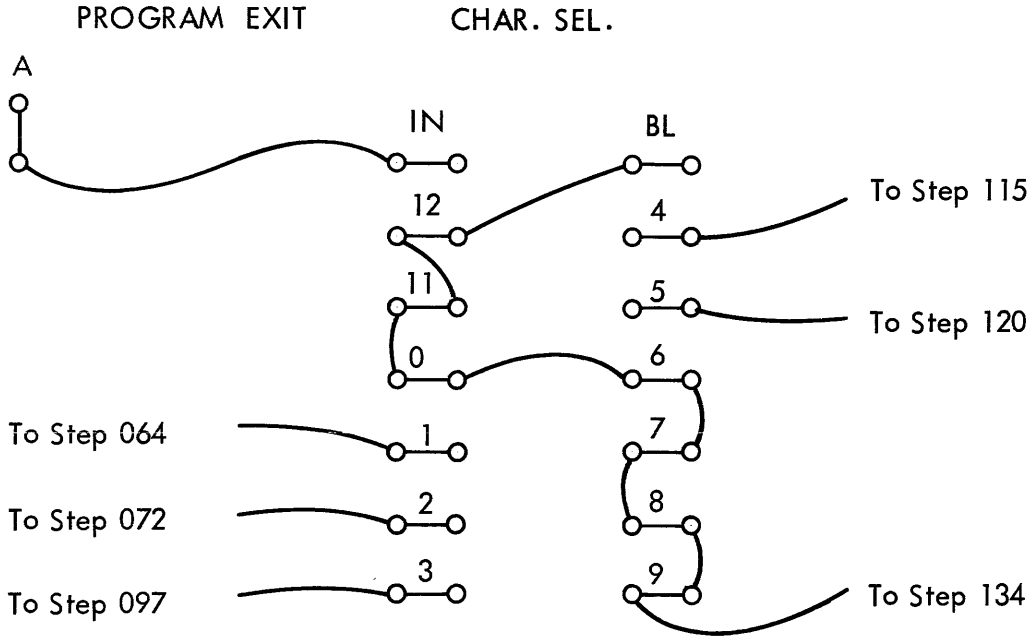


Figure 36

Another solution to this problem is found in the following method. In this method two additional program steps are required (60 milliseconds) and also a section each on the compare selector and on the accumulator sign section. Figure 37 illustrates the panel wiring and the program steps.

This method is based on the fact that the numerical digits, 1-9, are the only nine RAMAC characters that both appear positive when read into the accumulator and remain unchanged after passing through an accumulator units digit. Any character other than 1-9 will cause the impulse from Program Exit A to be diverted directly to step 134.

Prog. Step	FROM			TO			No. Char-acters		Control		Description
	Track	Position		Track	Position						
xxx	W	0	0	-	9	9	0	1			Store card code in character selector.
xxx	W	0	0	L	9	9	0	1		5	Reset-Add card code in accumulator 9
xxx	L	9	9	W	0	0	0	1	A	1	Compare accumulator with card code.

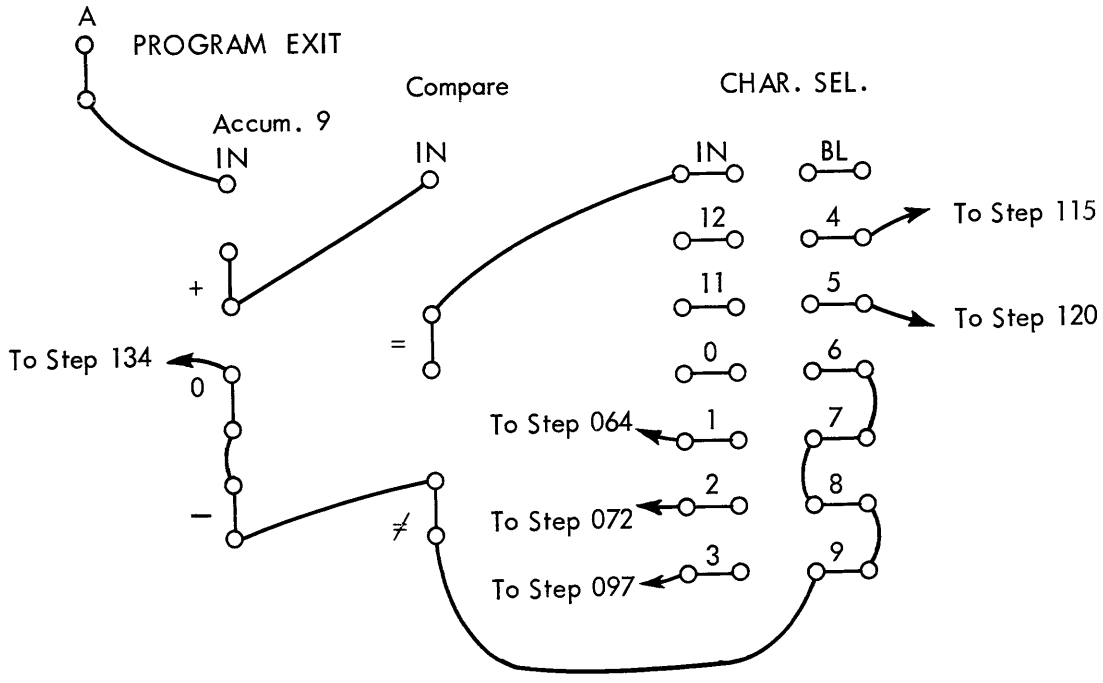


Figure 37

The economical use of control panel features implies minimizing the number of control panel components needed to initiate each function. The programmer should strive to obtain as much use as possible from each panel feature used with an aim toward eliminating redundant wiring. Usually, the over-all problem is subdivided into parts by considering each Exit as an individual problem. For example, Cycle Delay 8 Out, though closely related logically to the circuit that impulses the In hub, is considered along with its functions as a separate problem.

The most important technique in minimizing the number of control panel components is the assignment of step numbers that favor control-panel wiring (Figure 38). The best time to apply this technique is after the control panel function chart (page 81) has been drawn. This chart presents the logical branching requirements of the program in clear, compact form.

In Figure 38, Program Exit A is used to test accumulator 1. Each status hub of the accumulator is used to enter the program at different steps. If steps 048, 055, and 060 are used as entry points the wiring would require three double distributors. If, however, steps 042, 046, and 049 were selected as in Figure 38, two double distributors would be saved. Program Entries 059, 069, and 079 would have worked equally well.

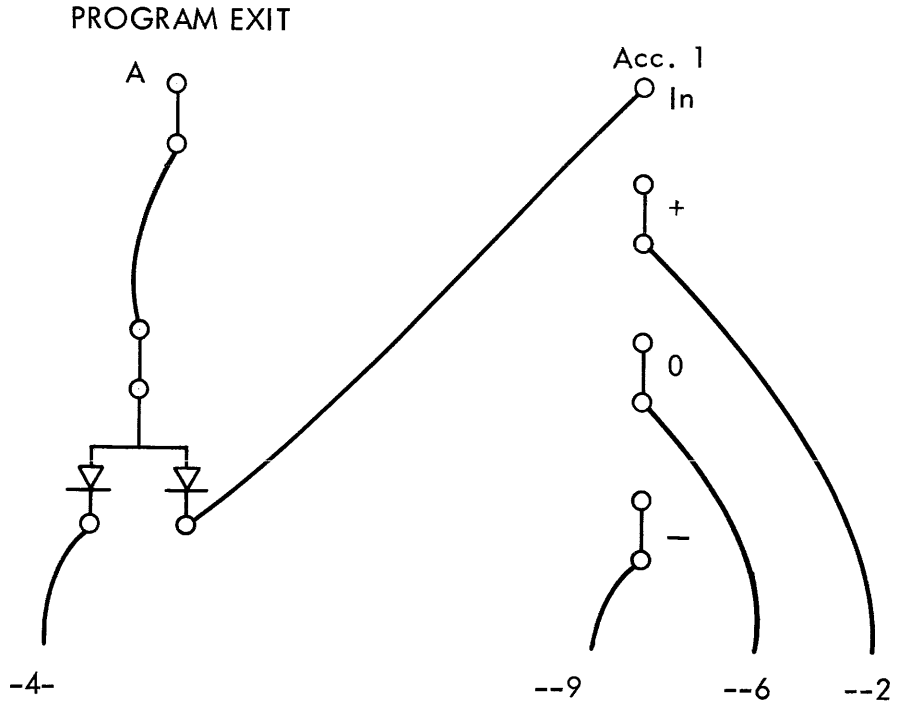


Figure 38

When two or more exit hubs are required to initiate similar functions, they should be considered as one wiring problem. In such cases, distributors and logical device sections may frequently be shared. This is illustrated in Figures 39 and 40.

In Figure 39, Program Exit B is used to test the compare selector. On an equal compare, a program advance occurs, and on an unequal compare the program transfers to step 092. Program Exit C is used to test the blank transmission. A "yes" on this selector causes a transfer to step 092 and a "no" causes a transfer to step 057. Each program exit can be wired separately using three double distributors. However, combining the program exits (Figure 39) saves one double distributor.

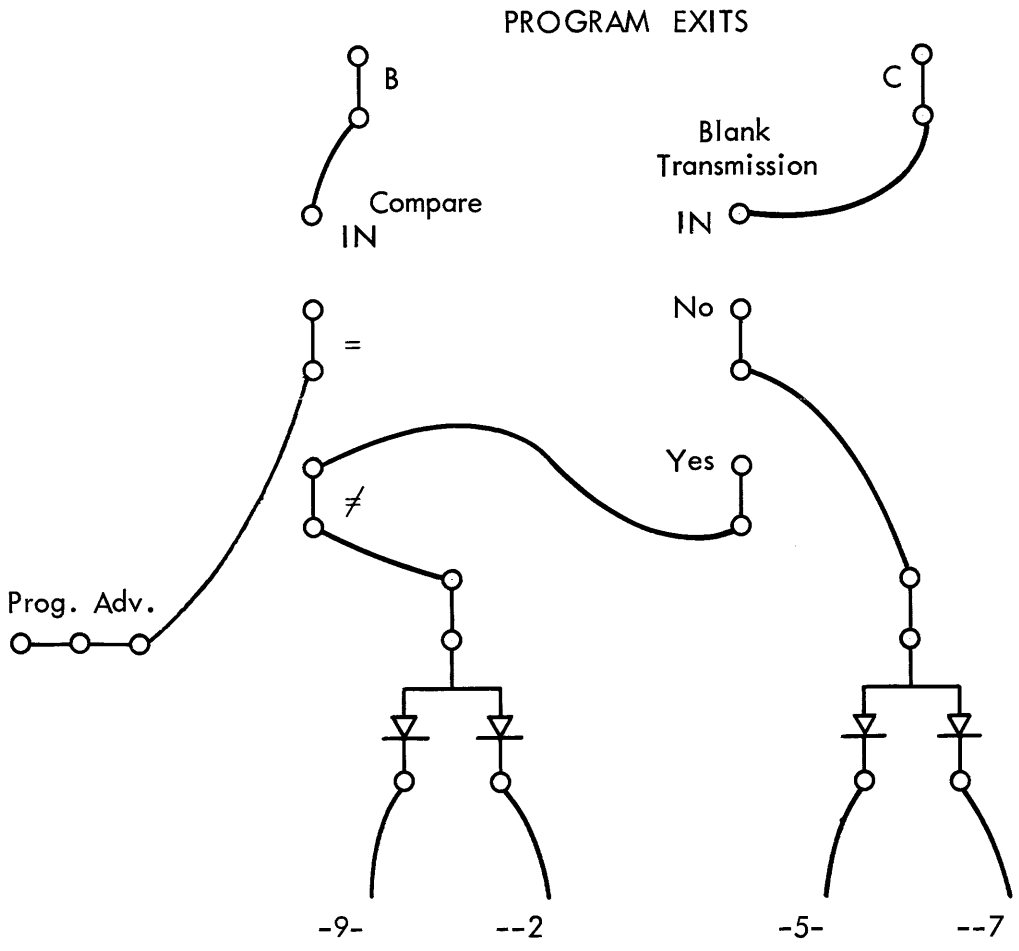


Figure 39

In Figure 40, Program Exits X and Y are to perform the functions diagrammed in the flow chart at the left. In the diagram showing the solution, both Program Exits were wired to the same section of Selector 1 rather than two separate sections.

Figures 39 and 40 illustrate two other considerations that are frequently disregarded in process panel wiring:

1. When an exit hub, under some set of circumstances, impulses only one type of function hub, a single distributor is not required in the path (Figure 39).

2. In wiring logical device sections, it is not mandatory that the Exit impulse enter at the In (or C) hub and leave by the status hubs. Wiring in the other direction can frequently save sections or distributors (Figure 40).

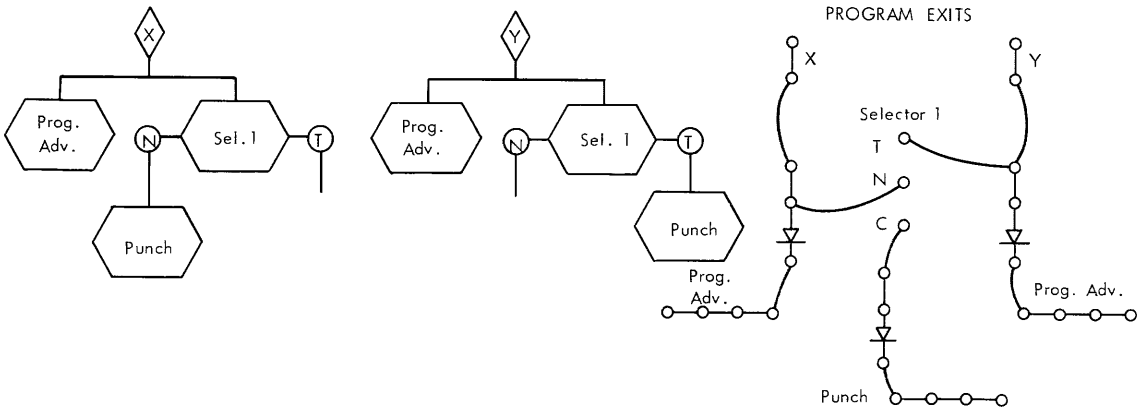


Figure 40

The programmer should be careful not to use one logical device when another would be preferable. In particular, the character selector is often used to excess. For example, if a program is to transfer to a subroutine, when the character in W99 is an asterisk, and to a program advance if it is any other character, perhaps the first instinct is to use the alphanumeric section of the character selector. Rather than connect the asterisk to the first step of the subroutine and all other status hubs to program advance, it is clearly preferable to store an asterisk in some constant area, X99 for instance, and use the compare selector to transfer the program. On the compare instruction, W99 X99 01 A1, Program Exit A tests the comparison and on the equal compare transfers to the subroutine. An unequal compare transfers to program advance.

It is usually better to wire logical sections in "series" rather than "parallel." This practice will reduce the possibility of back circuits. For example, Program Exit A is to transfer the program to step 040 if either Selector 1 or Selector 2 is transferred; otherwise to step 080 (Figure 41).

Figure 41 is incorrect, as is evident when one considers what happens when one selector is normal and the other is transferred. Figure 41B shows the correct solution.

It is characteristic of complex board wiring problems that many alternate solutions will obtain the same results. Many of these will be clearly uneconomical; of those that prove economical, it will be hard to choose the proper one. Of these solutions that are feasible to attempt, there will usually be two extremes.

The first solution requires many distributors, but few sections of the logical devices are involved. Frequently only one section of the device is necessary. The second solution requires many logical device sections, but few distributors.

There is, of course, the possibility of intermediate or "compromise" solutions existing. The programmer's choice must depend upon the panel components available. To arrive at the best solution, it is best to diagram the several possible solutions and, by comparison, select the best one.

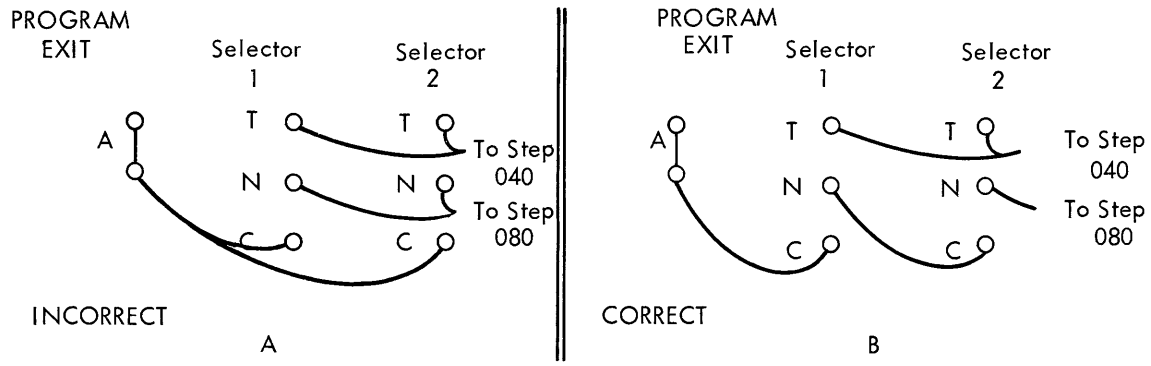


Figure 41

The first solution using many distributors but few logical device sections, can generally be obtained directly from the function flow chart by merely placing distributors in the wires that are connected to the function hubs (Figure 42).

As can be seen from Figure 42, the requirements are: one section of accumulator 1, three double distributors, and two single distributors.

The second solution, using few distributors but many logical device sections, is not obtained as naturally as the first solution. The results, however, can be obtained with ease by the use of an auxiliary diagram and a series of four steps. The notes enclosed in parentheses in the following list of the four steps, apply to the example explained in Figure 42.

1. Distribute the exit impulse into a number of branches equal to the maximum number of functions that must be impulsed (3).

2. Label the branches (Tens, Units, Selector Set).

3. In each branch enter a symbol for the logical devices that must control that branch (accumulator 1 in each branch).

4. With reference to the logical requirements, label the output levels and connect the logical devices back to the distributors, working from right to left.

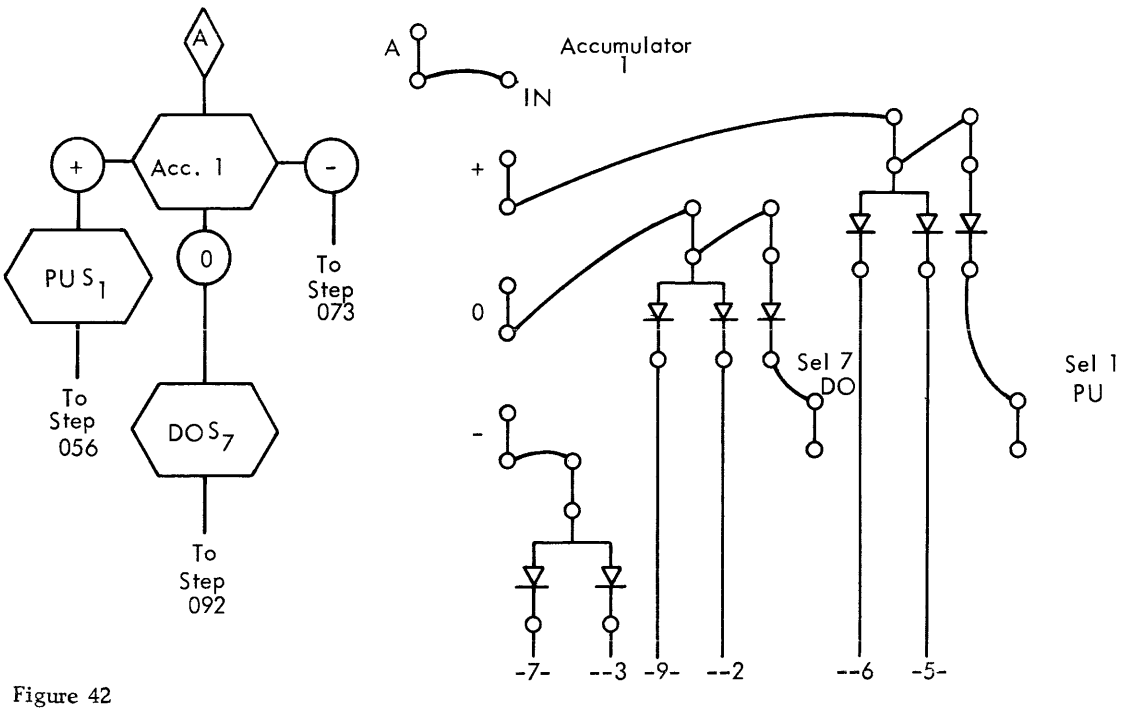


Figure 42

The auxiliary diagram and the resultant wiring diagram are shown in Figure 43.

The requirements for solution two are: three sections of accumulator 1, one double and one single distributor.

The auxiliary diagram can often be used to obtain the intermediate solutions for inspection. These solutions are obtained by combining two of the branches and replacing the logical element section(s) so saved with the required distributors. In Figure 43 this is done by combining the two right sections of accumulator 1, using a distributor to impulse Units 6 and the Pick Up of selector 1 with the impulse from Plus. The impulse from Zero is wired through through a distributor to Units 2 and Drop Out of selector 7. The impulse from Minus is wired through a single distributor to Units 3.

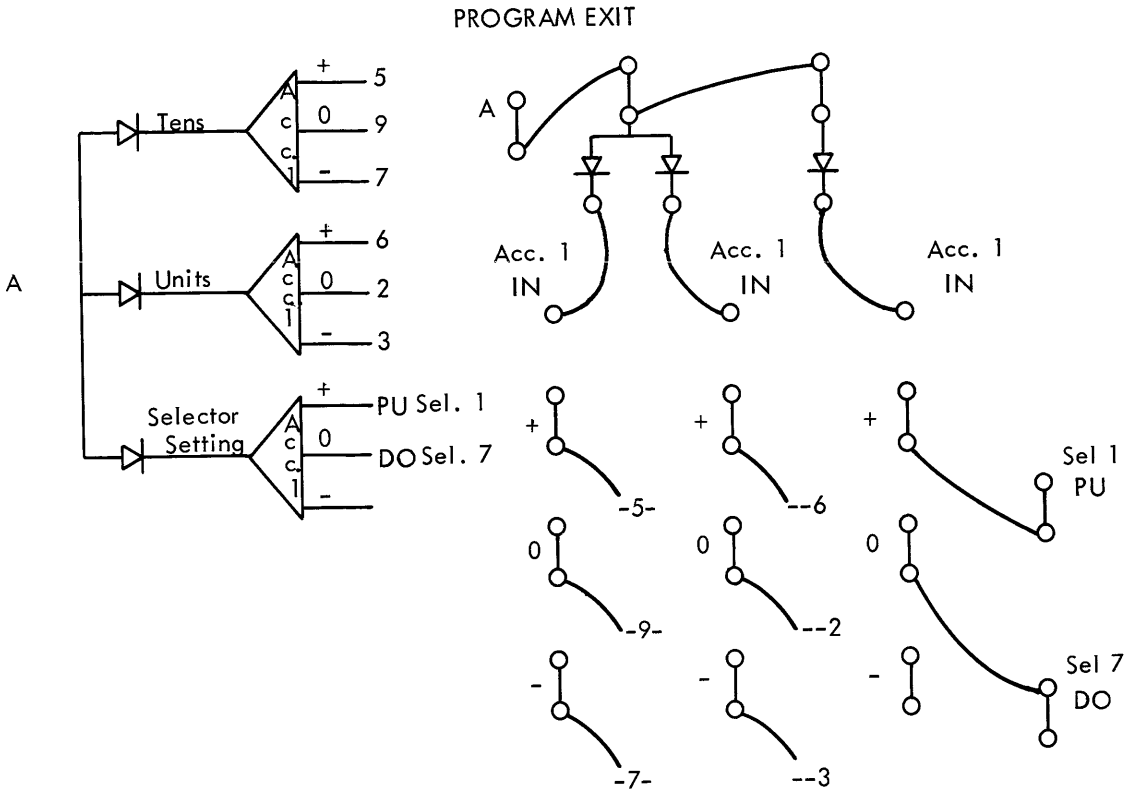


Figure 43

A more complex example is shown in Figure 44A. The first solution is not drawn because it can be immediately derived from Figure 44A. The requirements for solution one are: one section of accumulator 2, one section of accumulator 5, one section of selector 2, one section of selector 3 and six double distributors. Figure 44B is the auxiliary diagram for the second solution shown in Figure 45.

The requirements for the second solution are: two sections of accumulator 2, one section of accumulator 5, two sections of selector 2, one section of selector 3 and one double distributor. A close examination of Figure 45 reveals that the sections enclosed within the dotted lines are unnecessary, because Program Entry Tens 9 is to be impulsed in every case. Thus, if these diagrams can be drawn before step numbers are finally assigned, considerable economies may accrue.

There is a final consideration concerning the process panel, as well as the other RAMAC panels. An exit hub or distributor may be inadvertently subjected to an electrical overload by wiring too many function hubs to it. Although the ratings of these hubs are usually adequate, it is well to be aware of their limits. These are available in the form of rating charts for each machine panel (see Appendix C, Section 4).

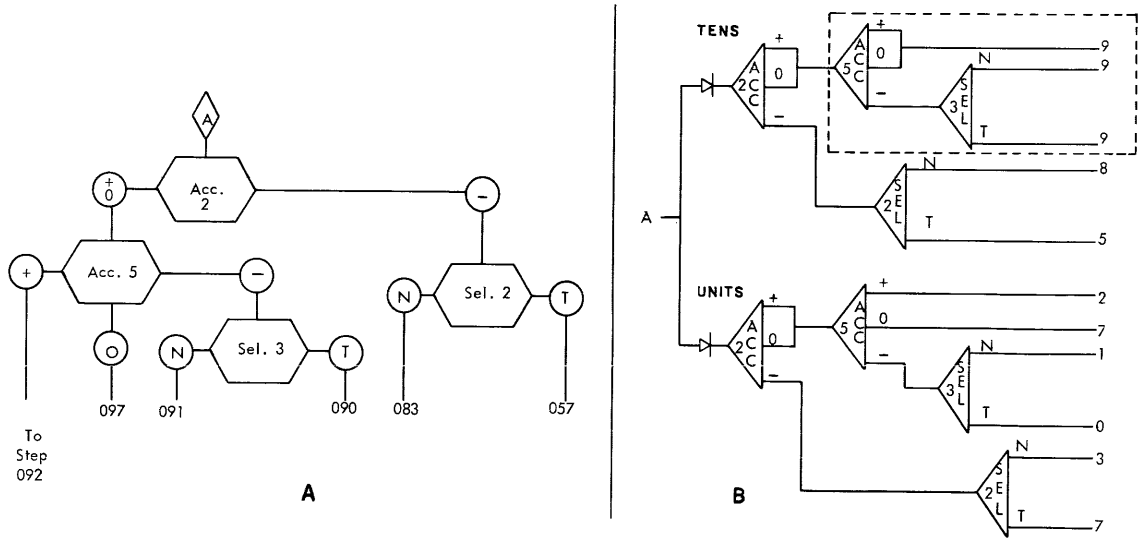


Figure 44

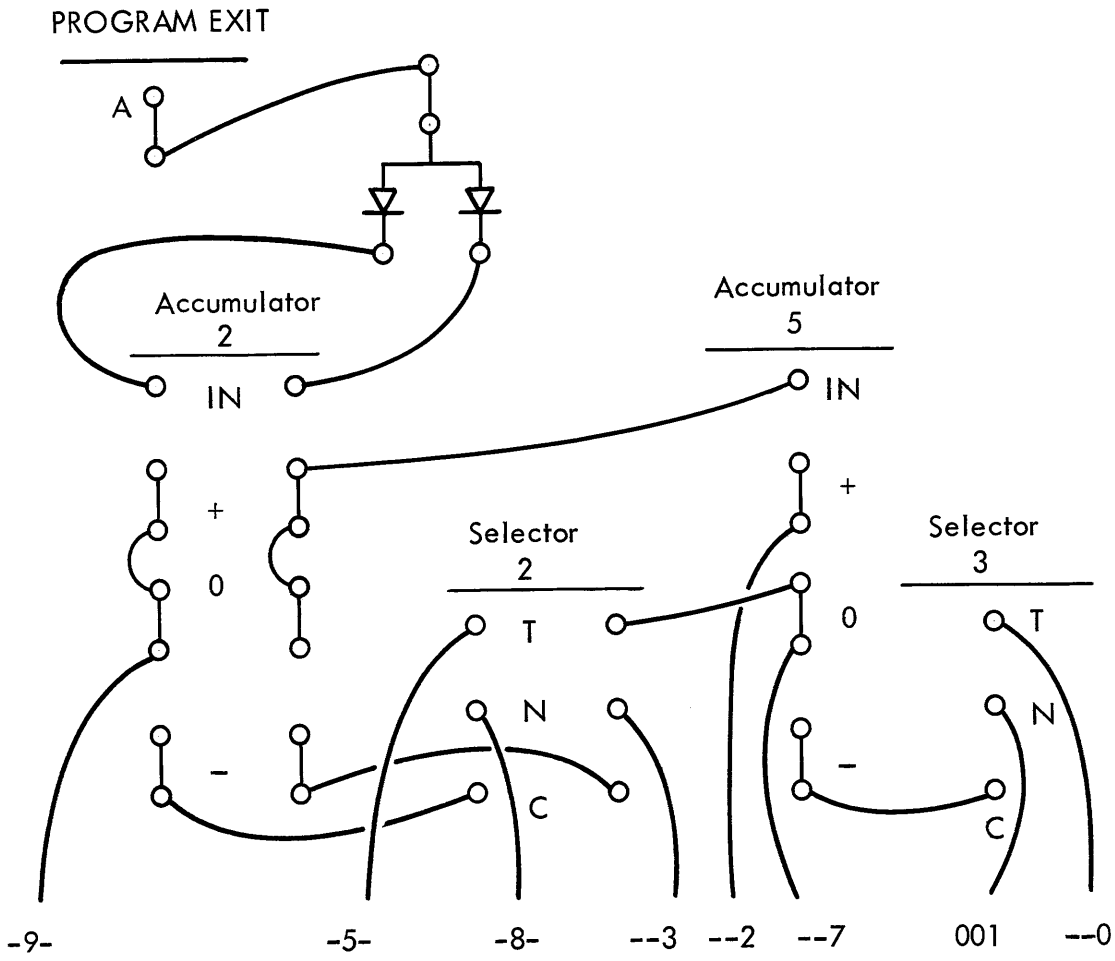


Figure 45

PROGRAMMED CHECKING

As outlined previously, accounting controls must be established to assure the proper functioning of the machine system and its related procedures. This is a major type of check, and it is advisable to incorporate certain other checks of a minor nature. The automatic checking performed in certain areas of the machine accomplishes a considerable check on data transfers; however, programmed checks are suggested to place a reasonable check on other areas of operation.

A check on the file access operation is one of the best over-all performance checks for the machine. This is particularly true if indirect addressing is used. A compare check between a file item identification number and the input item control data proves all machine operations performed during the process of locating a file item. This can be accomplished through one program step and 50 milliseconds of time.

It may be wise to include the access position check at the very beginning of operation. When the machine is received and placed in operation, a utility program can be used to record the file addresses in all 50,000 record locations. Then, on the initial file load, these addresses can be used for access compare-check. Whenever a record location is vacated, the address of that location should be retained in the record. Once this type of check is placed in use as a standard operation, there is no problem of processing an incorrect record.

Some type of accuracy check can be used on program load routines; there are several approaches to this check. A "hash" total check can be done by adding the ten instructions on a program track into the ten accumulators (on one step), and by then going through a series of accumulator folding operations to develop a ten-digit total. Each program track is likewise added to develop a ten-digit hash total that is compared for accuracy. The second method of using the hash total principle is to develop a total in one accumulator using several selected positions from one instruction per track. One such check uses track locations 60-63. Because position 63 includes the To address, the Alpha zone supplements the check as sign control on the accumulator.

A less comprehensive check merely compares a single character position on each program track against a corresponding check character.

There are three types of arithmetic checks that can be incorporated in any program. They are:

1. Proof-factor type. This is a common type of check used in some warehousing operations:

   Cost + Weight + Selling Price = Proof Factor

   Total Cost + Total Weight + Total Selling Price = (Proof Factor x Quantity)

   The proof-factor value is carried in the item record.

2. Multiplying by complement of multiplier and balancing.

Example:

Actual Value Multiplication	Complement Multiplication	Sum
8.25 (Hours)	8.25	6.4350
x .78 (Rate)	x .22	1.8150
6.4350 (Earnings)	1.8150	8.2500

3. The reverse--arithmetic method.

When a card is punched, plans should be made to punch out as much variable record data in the card as card capacity permits. If opening and closing balances are punched, these can be related to the transaction value for the purposes of "off-line" checking, or for audit and record reconstruction purposes.

Control totals should be developed on all input data values wherever possible. The 305-developed totals are compared to external control totals in much the same manner as with standard punched-card machines.

A method of document control is to serial number each document and add the numbers to develop a control total. The total developed should equal:

$$\text{Number of Documents} \times \frac{(\text{opening number} + \text{closing number})}{2}$$

If the totals do not balance, either some items are missing or some extras have found their way into the system. If only one item is missing, the difference of the totals is the actual serial number of the missing item.

SYSTEM SPEED

To obtain the desired processing speed of a machine installation, certain factors should be kept in mind. Perhaps the most common fault of programming is the attempt to create a program that is designed to handle all types of input.

A program that is written to expedite the transaction (or input card) that occurs most frequently should be favored. The shortest and fastest program possible should be developed for this "main-line" item. This program should include only those test points, data transfers, etc., that are essential to processing this item, even if this approach tends to slow down the minor transaction routines.

The minor transactions generally account for only a small percentage of the total input, but may create many extra programming problems. There is a point beyond which it is not desirable to burden the entire program with a few exception items. When this condition exists, a study should be made to determine if the items should be processed separately to permit maximum program efficiency on the volume transactions.

Increasing the stored program speed by the use of block transfers and multiple transfers cannot be overemphasized. These effective types of instructions are the result of careful format design.

An important speed-increasing technique, with respect to the process control panel, is to design each program exit to perform as much work as possible. Where feasible, logical elements should be set up in groups, perhaps by several instructions; then they can all be tested with a single program exit. The time saved by eliminating a single program exit from a program can result in significant speed increases. For example, in a program that will produce 60 output cards per minute, one less program exit can increase the program speed by 2 percent.

Planning input, output, and file-seeking operations, so that the other units of the RAMAC are kept in use during the time these mechanical units are in operation is extremely important. Although various machine units such as card reader, punch, printer, access mechanism, etc., can be operated simultaneously, they cannot operate completely parallel because information must flow from one unit to the next. However, the program should be planned to have all units operating as independently and continuously as possible. The printer and punch may be producing output for card 1, while the processing and file areas are processing card 2, and the reader is reading card 3. It can now be seen that, to keep all units in operation and to develop maximum system speed, the time cycle for each area must be approximately equal.

The K, S, and Q tracks act as buffers for the reader, punch/printer, and typewriter, respectively. Processing can proceed independently of these units as long as processing reference is not made to one of these three tracks, at a time when the associated unit is in operation. When this does occur, processing will be interlocked (i. e., the instruction will not be executed) until the input (or output) cycle terminates.

One other very important speed-limiting condition is the use of the typewriter for more than an occasional exception type-out. The typewriter presents no significant delay except on successive programmed type-outs. Then the entire

machine system is held up until the first type-out is completed so that the second message can be transferred to the Q track for typing. Therefore, if programmed type-out is being considered, it should be determined how often the type-outs will occur and what effect this will have on the over-all system speed. If programmed type-outs are used, another point to keep in mind is the addition of the address keying time on manual inquiry.

OPERATING THE 305

MANUAL INQUIRY

The console typewriter is intended primarily for inquiry print-out and for operator communication with the machine. It may also be used for a programmed exception print-out. Frequent exception print-outs are not recommended because this reduces the speed of the entire machine system to that of the typewriter.

When an inquiry is made with the ITI hubs not wired, the 305 processing program is not affected until the complete five-digit file address is keyed in. The file address is also typed on the keying operation. The 305 is interlocked on a manual inquiry for approximately 150 milliseconds, plus the file access time to locate the desired record.

A small control panel is provided for typing format control. Three basic formats are provided for inquiry print-out. Additional format variations may be programmed on the control panel by analysis of control characters on the Q track.

A cross-reference index is required for manual inquiry when an indirect address system is used. At the time the file is loaded, cards should be punched containing the control data and the internal machine address. These can be used to produce the index.

An inquiry operation can be performed from punched input cards. A card code is assigned and a short, processing routine developed to process an inquiry. Because the standard programs for address conversion are used, the cross-reference index is not required for card inquiry. The machine is programmed to create whatever type of output reply is desired. If the typewriter is used for print-out, the format is controlled in the same way as any other programmed print-out.

When the typewriter is used for <u>any</u> programmed print-out, the ITI hub on the 305 panel must be plugged if manual inquiry also will be used. This prevents a conflict of use between the program and a manual inquiry. Otherwise, the operator could be keying-in (and typing) a manual inquiry address at the same time the machine found the need for a programmed print-out. With the ITI plugged, the operator depresses one of the three format keys to signal a request for inquiry. Then, the operator must wait until the inquiry light signals that

the machine has reached the inquiry point provided in the program. Now the machine is interlocked while the five-digit file address is keyed and the file record located. Thus, keying time is added to manual inquiry time if the ITI interlock is plugged. An incomplete keying operation will interlock the machine indefinitely when ITI is plugged.

TEST DATA

Plans should be made to run test transactions through the machine each time after the power is first turned on, and whenever it is set up to run a different program. The test situation should use all areas of the machine and test as many conditions in the program as possible. Pre-calculated values should be established to check arithmetic results, card reading, punching, printing, and so forth. The test should be designed to use at least one instruction from each program track and refer to all other drum tracks. The end of the test should clear all areas ready for actual processing.

CONSOLE OPERATION

The console manipulation of the 305 allows the operator to alter source data. Also, it is possible to modify program instructions or, in some other manner, change the result of the program. This type of operation is necessary for an in-line data processing machine. However, a problem could develop if these operations are not executed in the proper manner. Therefore, it is imperative that console procedures be developed and program information be provided that will reduce the operator's function to a series of planned operations.

No program condition that can properly be done by internal programming should stop the machine and require operator attention. The program should be designed with a series of planned re-start points. If some program difficulty is encountered along the way, it should only be necessary for the operator to return the program to the last-passed re-start point to resume operations.

Situations probably will develop where human decision is required. It would be advisable to have the programmer or supervisor take care of these conditions. Even so, the person who handles these will need a thorough knowledge of the machine operation and complete program information, as well as a clear understanding of the console controls.

The program data must be complete and kept up to date. The following documents should be available to the console operator:

1. A complete detailed listing of the program instructions; IBM Form X26-6328 or X26-6343.

2. A complete set of track layout diagrams; IBM Form X26-6269.

3. A wiring diagram for each 305 control panel program exit; IBM Form X26-6343.

4. A listing of all selectors, their purpose, pickup and drop points and where used; IBM Form X26-7502.

5. A 305 control panel function chart.

6. Detailed wiring diagrams of the 323, 370, and 380 control panels; IBM Form X26-6329.

7. A detailed block diagram of the program.

8. A 370 planning and spacing chart; IBM Form X26-0636.

MISCELLANEOUS

COMMON MISTAKES

The following are some of the most common mistakes made during the course of programming or operating the 305 RAMAC:

1. Wasting machine time by processing on tracks K, S, and Q, or by not properly overlapping the performance of the mechanical units.

2. Failing to initialize machine components before use; i. e., not blanking the drum tracks when initially necessary, not resetting a selector before testing or not dropping out the accumulator overflow selector before testing, etc.

3. Using an R or L in the To portion of a compare instruction.

4. Comparing with data on track K, during the processing of the last card or single cards.

5. Failing to reset accumulators 0 and 1 before multiplying.

6. Planning inquiry for the wrong place in the program.

7. Making changes in the machine program, without updating data sheets, program cards, etc.

8. Not identifying modified and modifying instructions to enable easy manual reference and alteration.

9. Taking constant data out of instruction. If instructions change relative position, wrong constant data will be used.

10. Using machine components through communication channels without due regard.

11. Using other than (b)99 to blank a track.

12. Reading into other than position 99 of track V.

13. Debugging program with file interlock plugged. This allows data to be written in the file.

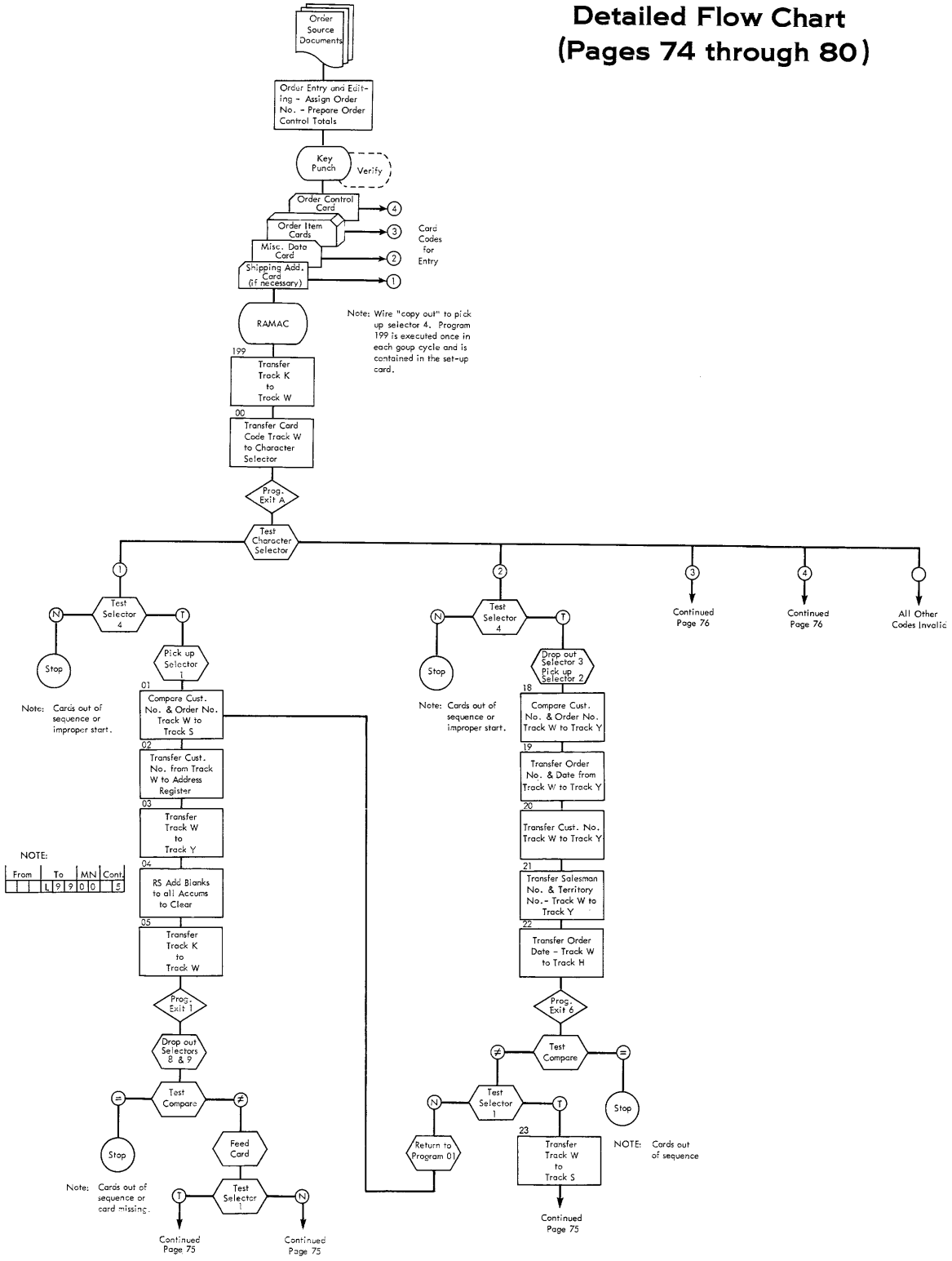
14. Not setting alteration switches correctly.

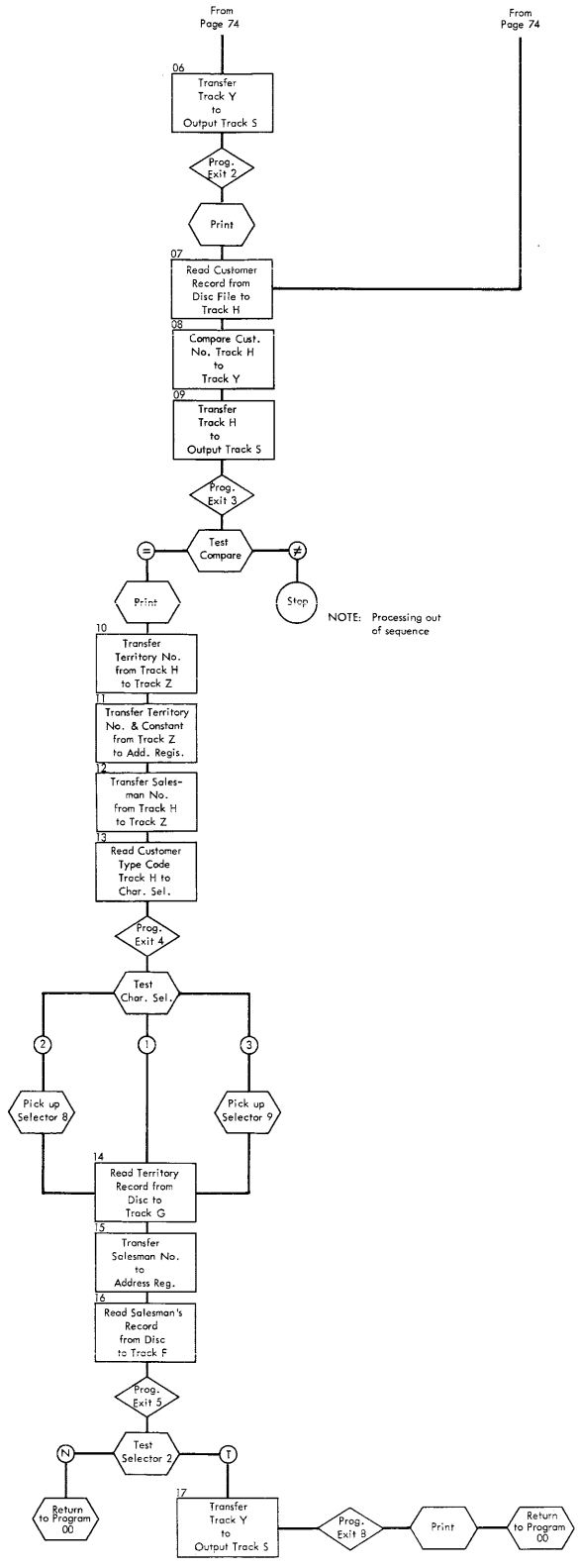
15. Leaving the test lock on inadvertently.

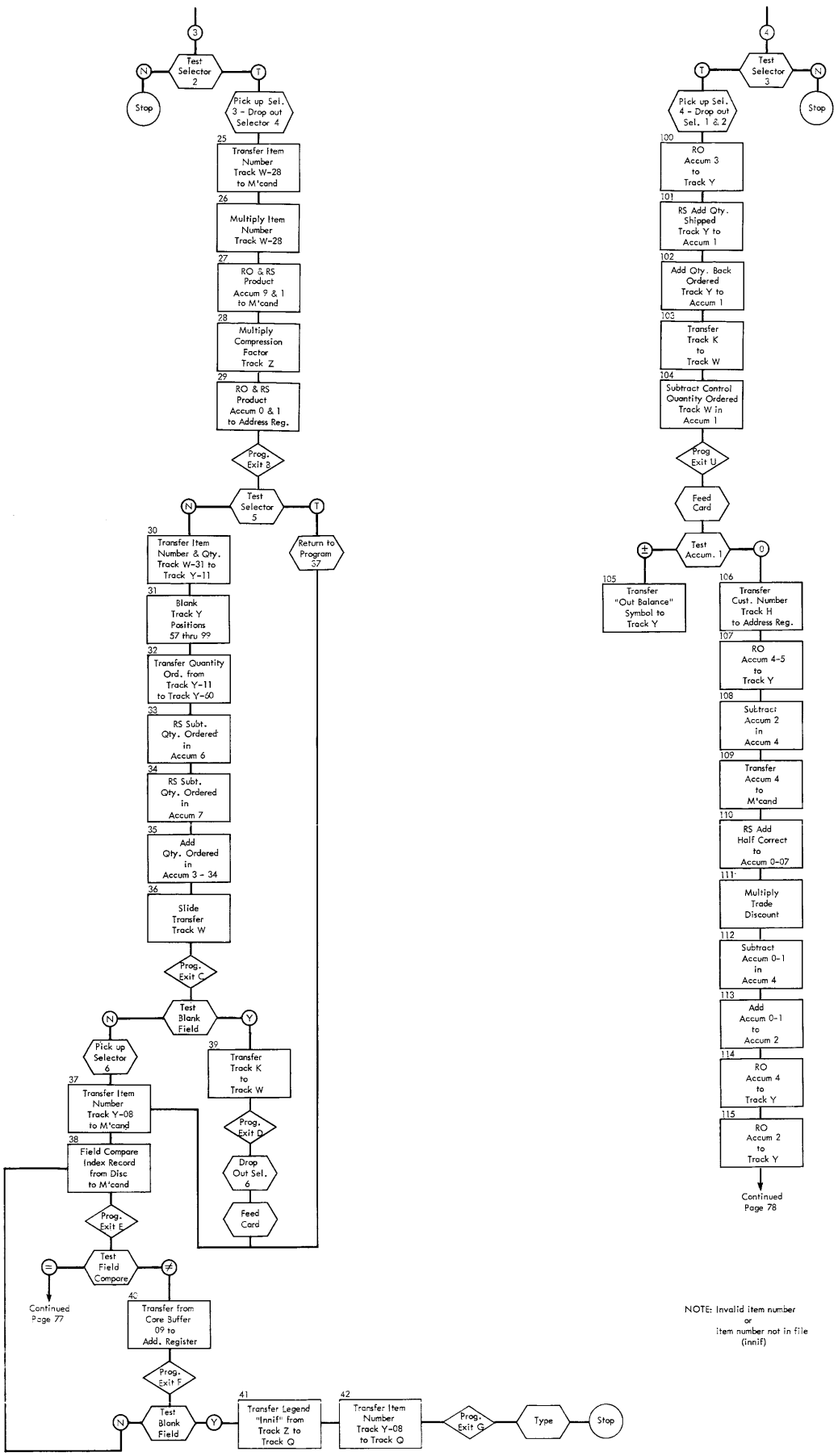
DOCUMENTATION

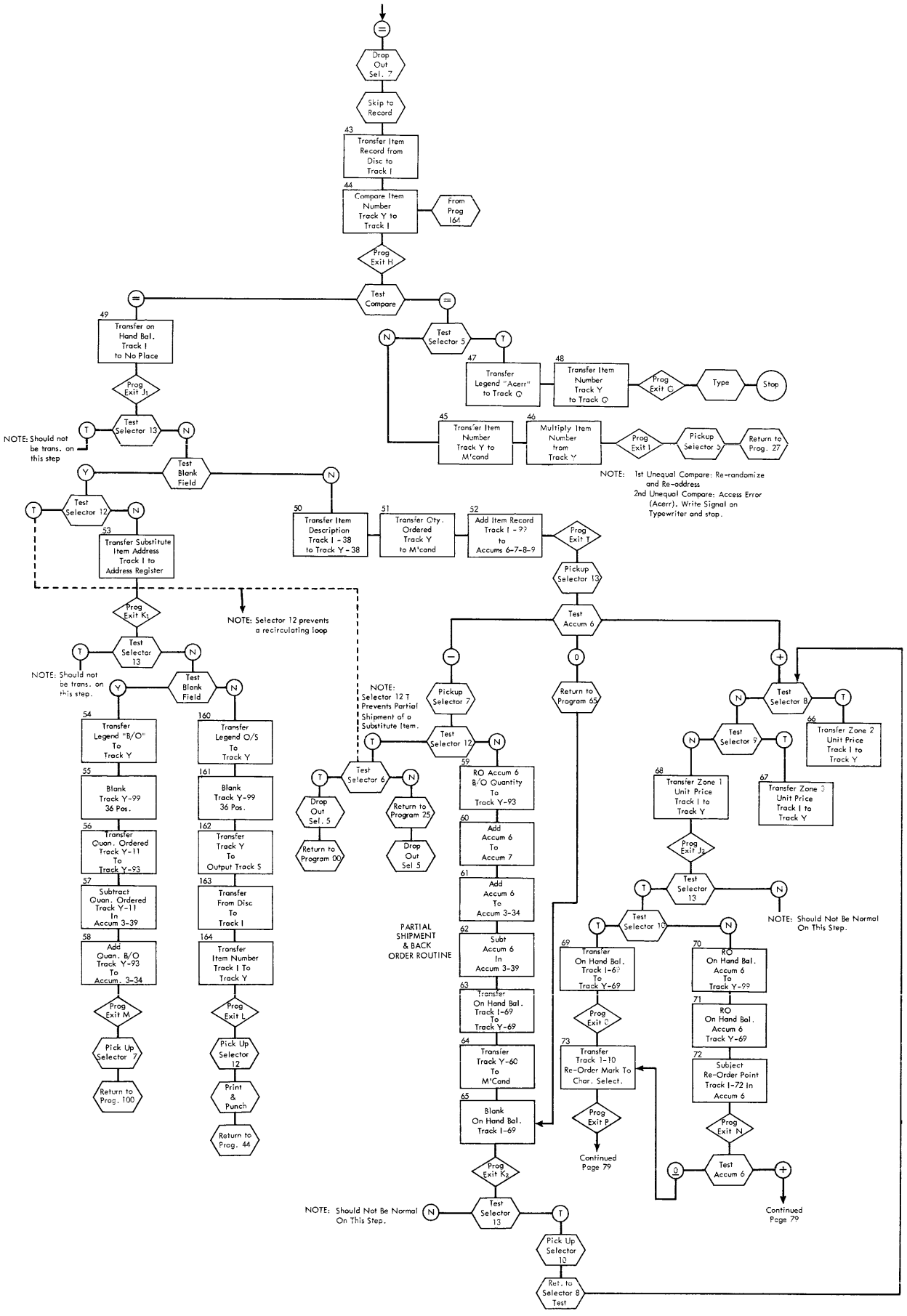
The following pages illustrate the documents mentioned in the section, Programming Practice, page 27:

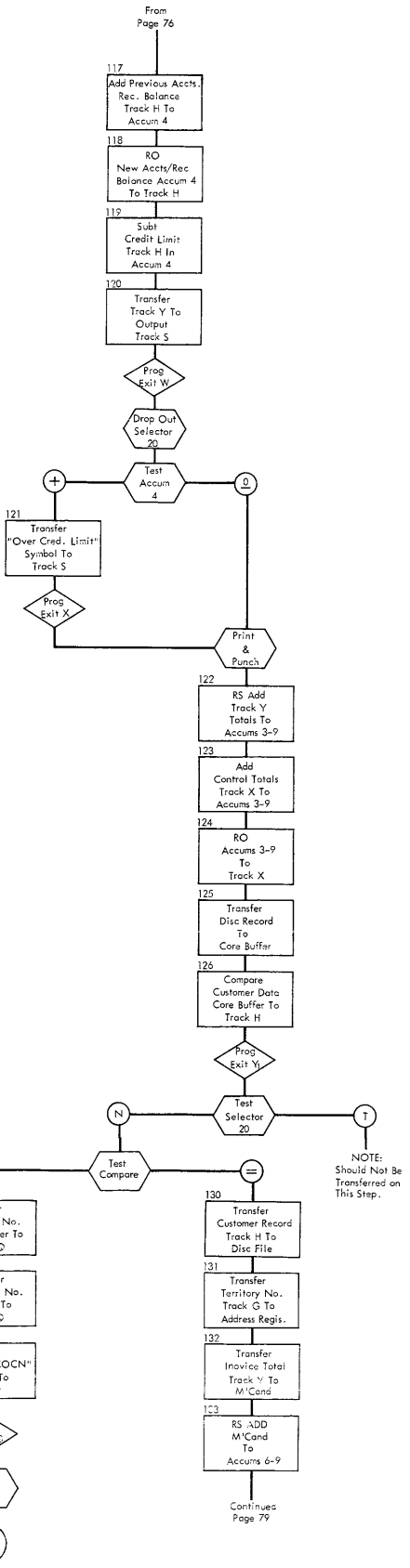
Detailed Flow Chart (Pages 74 through 80)

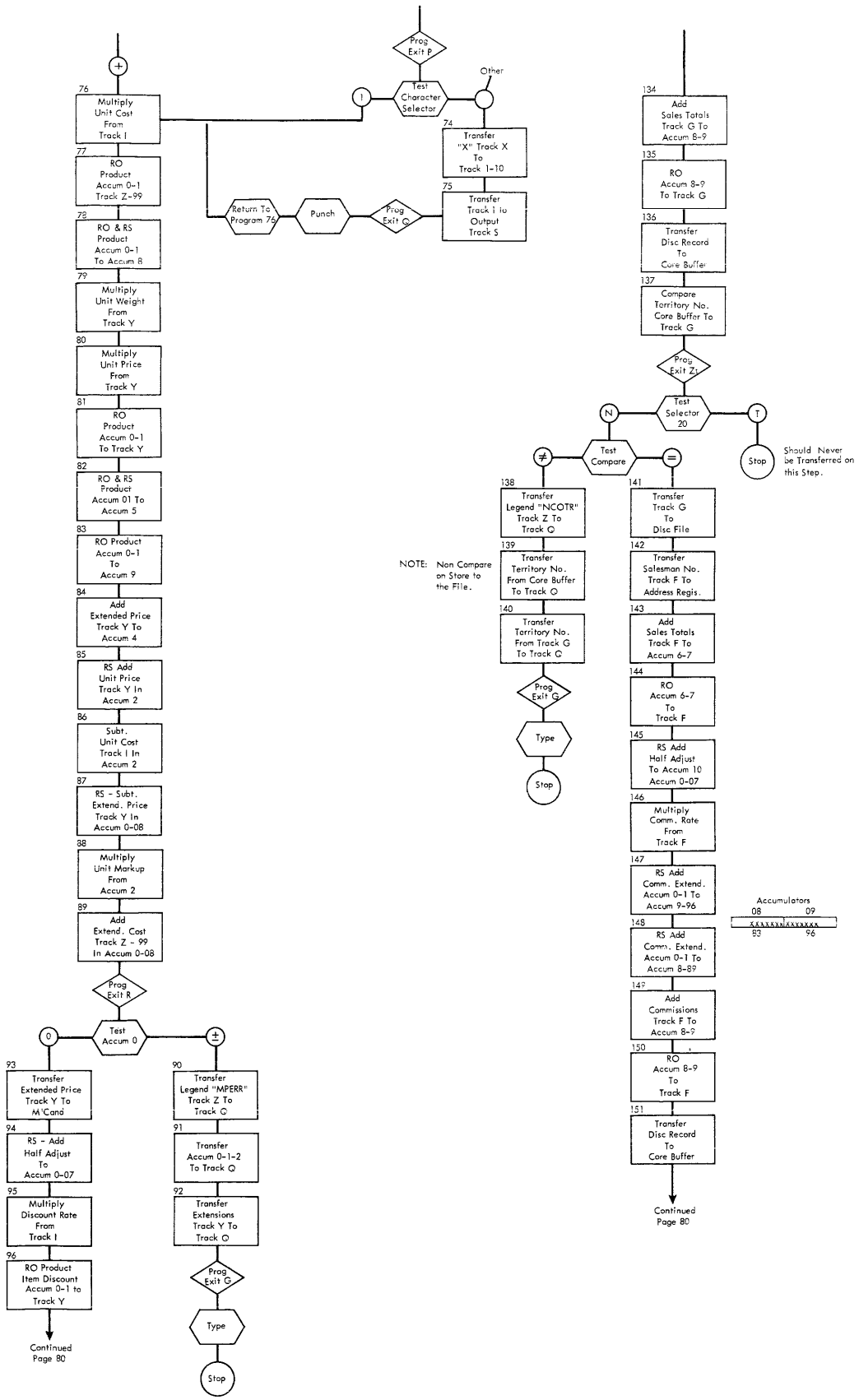


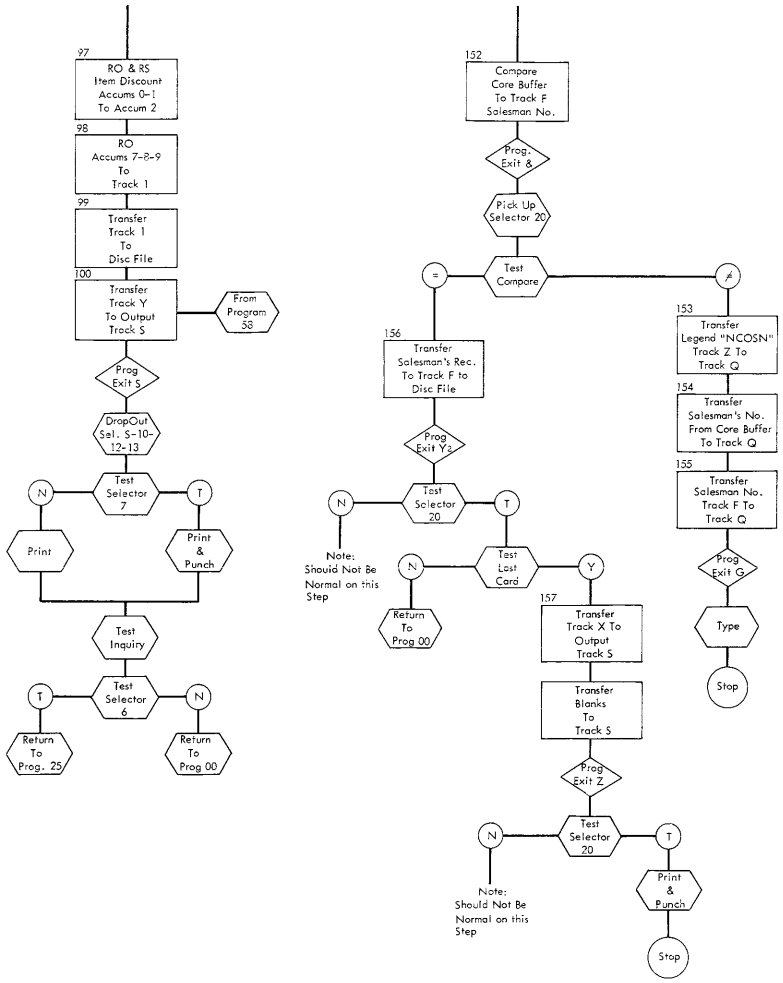

From Page 74
From Page 74
From Page 74
06
Transfer Track Y to Output Track S
Prog. Exit 2
Print
07
Read Customer Record from Disc File to Track H
08
Compare Cust. No. Track H to Track Y
09
Transfer Track H to Output Track S
Prog. Exit 3
Test Compare
=
≠
Print
Stop
NOTE: Processing out of sequence
10
Transfer Territory No. from Track H to Track Z
11
Transfer Territory No. & Constant from Track Z to Add. Regis.
12
Transfer Salesman No. from Track H to Track Z
13
Read Customer Type Code Track H to Char. Sel.
Prog. Exit 4
Test Char. Sel.
2
1
3
Pick up Selector 8
Pick up Selector 9
14
Read Territory Record from Disc to Track G
15
Transfer Salesman No. to Address Reg.
16
Read Salesman's Record from Disc to Track F
Prog. Exit 5
Test Selector 2
N
T
Return to Program 00
17
Transfer Track Y to Output Track S
Prog. Exit 8
Print
Return to Program 00
24
Transfer Track K to Track W
Prog. Exit 7
Print Feed Card
Return to Program 00


3
Test Selector 2
N
T
Stop
Pick up Sel. 3 - Drop out Selector 4
25 Transfer Item Number Track W-28 to M'cand
26 Multiply Item Number Track W-28
27 RO & RS Product Accum 9 & 1 to M'cand
28 Multiply Compression Factor Track Z
29 RO & RS Product Accum 0 & 1 to Address Reg.
Prog. Exit B
Test Selector 5
Return to Program 37
30 Transfer Item Number & Qty. Track W-31 to Track Y-11
31 Blank Track Y Positions 57 thru 99
32 Transfer Quantity Ord. from Track Y-11 to Track Y-60
33 RS Subt. Qty. Ordered in Accum 6
34 RS Subt. Qty. Ordered in Accum 7
35 Add Qty. Ordered in Accum 3 - 34
36 Slide Transfer Track W
Prog. Exit C
Test Blank Field
Y
39 Transfer Track K to Track W
Prog. Exit D
Drop Out Sel. 6
Feed Card
Pick up Selector 6
37 Transfer Item Number Track Y-08 to M'cand
38 Field Compare Index Record from Disc to M'cand
Prog. Exit E
Test Field Compare
=
≠
Continued Page 77
40 Transfer from Core Buffer 09 to Add. Register
Prog. Exit F
Test Blank Field
Y
41 Transfer Legend "Innif" from Track Z to Track Q
42 Transfer Item Number Track Y-08 to Track Q
Prog. Exit G
Type
Stop
4
Test Selector 3
T
N
Stop
Pick up Sel. 4 - Drop out Sel. 1 & 2
100 RO Accum 3 to Track Y
101 RS Add Qty. Shipped Track Y to Accum 1
102 Add Qty. Back Ordered Track Y to Accum 1
103 Transfer Track K to Track W
104 Subtract Control Quantity Ordered Track W in Accum 1
Prog Exit U
Feed Card
Test Accum. 1
±
0
105 Transfer "Out Balance" Symbol to Track Y
106 Transfer Cust. Number Track H to Address Reg.
107 RO Accum 4-5 to Track Y
108 Subtract Accum 2 in Accum 4
109 Transfer Accum 4 to M'cand
110 RS Add Half Correct to Accum 0-07
111 Multiply Trade Discount
112 Subtract Accum 0-1 in Accum 4
113 Add Accum 0-1 to Accum 2
114 RO Accum 4 to Track Y
115 RO Accum 2 to Track Y
Continued Page 78
NOTE: Invalid item number or item number not in file (innif)


Drop Out Sel. 7
Skip to Record
43 Transfer Item Record from Disc to Track I
44 Compare Item Number Track Y to Track I
From Prog 164
Prog Exit H
Test Compare
49 Transfer on Hand Bal. Track I to No Place
Prog Exit J1
Test Selector 13
NOTE: Should not be trans. on this step
Test Blank Field
Test Selector 12
53 Transfer Substitute Item Address Track I to Address Register
Prog Exit K1
Test Selector 13
NOTE: Should not be trans. on this step.
Test Blank Field
54 Transfer Legend "B/O" To Track Y
55 Blank Track Y-99 36 Pos.
56 Transfer Quan. Ordered Track Y-11 To Track Y-93
57 Subtract Quan. Ordered Track Y-11 In Accum 3-39
58 Add Quan. B/O Track Y-93 To Accum. 3-34
Prog Exit M
Pick Up Selector 7
Return to Prog. 100
160 Transfer Legend O/S To Track Y
161 Blank Track Y-99 36 Pos.
162 Transfer Track Y To Output Track S
163 Transfer From Disc To Track I
164 Transfer Item Number Track I To Track Y
Prog Exit L
Pick Up Selector 12
Print & Punch
Return to Prog. 44
NOTE: Selector 12 prevents a recirculating loop
50 Transfer Item Description Track I -38 to Track Y-38
51 Transfer Qty. Ordered Track Y to M'cand
52 Add Item Record Track I - 99 to Accums 6-7-8-9
Prog Exit T
Pickup Selector 13
Test Accum 6
NOTE: Selector 12 T Prevents Partial Shipment of a Substitute Item.
Pickup Selector 7
Test Selector 12
Test Selector 6
Drop Out Sel. 5
Return to Program 00
Return to Program 25
Drop Out Sel 5
59 RO Accum 6 B/O Quantity To Track Y-93
60 Add Accum 6 To Accum 7
61 Add Accum 6 To Accum 3-34
62 Subt Accum 6 In Accum 3-39
63 Transfer On Hand Bal. Track I-69 To Track Y-69
64 Transfer Track Y-60 To M'Cand
65 Blank On Hand Bal. Track I-69
Prog Exit K2
NOTE: Should Not Be Normal On This Step.
Test Selector 13
Pick Up Selector 10
Ret. to Selector 8 Test
PARTIAL SHIPMENT & BACK ORDER ROUTINE
Return to Program 65
Test Selector 8
66 Transfer Zone 2 Unit Price Track I to Track Y
Test Selector 9
67 Transfer Zone 3 Unit Price Track I to Track Y
68 Transfer Zone 1 Unit Price Track I to Track Y
Prog Exit J2
Test Selector 13
NOTE: Should Not Be Normal On This Step.
Test Selector 10
69 Transfer On Hand Bal. Track I-69 To Track Y-69
70 RO On Hand Bal. Accum 6 To Track Y-99
Prog Exit O
71 RO On Hand Bal. Accum 6 Track Y-69
73 Transfer Track I-10 Re-Order Mark To Char. Select.
72 Subject Re-Order Point Track I-72 In Accum 6
Prog Exit P
Prog Exit N
Continued Page 79
Test Accum 6
Continued Page 79
Drop Out Sel. 7
Skip to Record
43 Transfer Item Record from Disc to Track I
44 Compare Item Number Track Y to Track I
From Prog 164
Prog Exit H
Test Compare
Test Selector 5
47 Transfer Legend "Acerr" to Track Q
48 Transfer Item Number Track Y to Track Q
Prog Exit Q
Type
Stop
45 Transfer Item Number Track Y to M'cand
46 Multiply Item Number from Track Y
Prog Exit I
Pickup Selector 5
Return to Prog. 27
NOTE: 1st Unequal Compare: Re-randomize and Re-address
2nd Unequal Compare: Access Error (Acerr). Write Signal on Typewriter and stop.


From Page 76
117
Add Previous Accts. Rec. Balance Track H To Accum 4
118
RO New Accts/Rec Balance Accum 4 To Track H
119
Subt Credit Limit Track H In Accum 4
120
Transfer Track Y To Output Track S
Prog Exit W
Drop Out Selector 20
Test Accum 4
+
0
121
Transfer "Over Cred. Limit" Symbol To Track S
Prog Exit X
Print & Punch
122
RS Add Track Y Totals To Accums 3-9
123
Add Control Totals Track X To Accums 3-9
124
RO Accums 3-9 To Track X
125
Transfer Disc Record To Core Buffer
126
Compare Customer Data Core Buffer To Track H
Prog Exit Y1
Test Selector 20
N
T
NOTE: Should Not Be Transferred on This Step.
Test Compare
≠
=
127
Transfer Customer No. Core Buffer To Track Q
128
Transfer Customer No. Track H To Track C
129
Transfer Legend "NCOCN" Track Z To Track C
Prog Exit C
Type
Stop
130
Transfer Customer Record Track H To Disc File
131
Transfer Territory No. Track G To Address Regis.
132
Transfer Inovice Total Track Y To M'Cand
133
RS ADD M'Cand To Accums 6-9
Continues Page 79


Prog Exit P
Other
1
Test Character Selector
74
Transfer "X" Track X To Track 1-10
75
Transfer Track 1 to Output Track S
Return To Program 76
Punch
Prog Exit Q
76
Multiply Unit Cost From Track I
77
RO Product Accum 0-1 Track Z-99
78
RO & RS Product Accum 0-1 To Accum 8
79
Multiply Unit Weight From Track Y
80
Multiply Unit Price From Track Y
81
RO Product Accum 0-1 To Track Y
82
RO & RS Product Accum 01 To Accum 5
83
RO Product Accum 0-1 To Accum 9
84
Add Extended Price Track Y To Accum 4
85
RS Add Unit Price Track Y In Accum 2
86
Subt. Unit Cost Track I In Accum 2
87
RS - Subt. Extend. Price Track Y In Accum 0-08
88
Multiply Unit Markup From Accum 2
89
Add Extend. Cost Track Z - 99 In Accum 0-08
Prog Exit R
Test Accum 0
0
±
93
Transfer Extended Price Track Y To M'Cand
94
RS - Add Half Adjust To Accum 0-07
95
Multiply Discount Rate From Track I
96
RO Product Item Discount Accum 0-1 to Track Y
Continued Page 80
90
Transfer Legend "MPERR" Track Z To Track Q
91
Transfer Accum 0-1-2 To Track Q
92
Transfer Extensions Track Y To Track Q
Prog Exit G
Type
Stop
134
Add Sales Totals Track G To Accum 8-9
135
RO Accum 8-9 To Track G
136
Transfer Disc Record To Core Buffer
137
Compare Territory No. Core Buffer To Track G
Prog Exit Z1
N
Test Selector 20
T
Stop
Should Never be Transferred on this Step.
≠
Test Compare
=
NOTE: Non Compare on Store to the File.
138
Transfer Legend "NCOTR" Track Z To Track Q
139
Transfer Territory No. From Core Buffer To Track Q
140
Transfer Territory No. From Track G To Track Q
Prog Exit G
Type
Stop
141
Transfer Track G To Disc File
142
Transfer Salesman No. Track F To Address Regis.
143
Add Sales Totals Track F To Accum 6-7
144
RO Accum 6-7 To Track F
145
RS Add Half Adjust To Accum 10 Accum 0-07
146
Multiply Comm. Rate From Track F
147
RS Add Comm. Extend. Accum 0-1 To Accum 9-96
148
RS Add Comm. Extend. Accum 0-1 To Accum 8-89
149
Add Commissions Track F To Accum 8-9
150
RO Accum 8-9 To Track F
151
Transfer Disc Record To Core Buffer
Continued Page 80
Accumulators
08
09
83
96


97
RO & RS
Item Discount
Accums 0-1
To Accum 2
98
RO
Accums 7-8-9
To
Track 1
99
Transfer
Track 1
To
Disc File
100
Transfer
Track Y
To Output
Track S
From
Program
58
Prog
Exit S
DropOut
Sel. S-10-
12-13
N
Test
Selector
7
T
Print
Print
&
Punch
Test
Inquiry
T
Test
Selector
6
N
Return
To
Prog. 25
Return
To
Prog 00
132
Compare
Core Buffer
To Track F
Salesman No.
Prog.
Exit &
Pick Up
Selector 20
=
Test
Compare
≠
156
Transfer
Salesman's Rec.
To Track F to
Disc File
Prog
Exit Y2
N
Test
Selector
20
T
Note:
Should Not Be
Normal on this
Step
N
Test
Last
Card
Y
Return
To
Prog 00
157
Transfer
Track X To
Output
Track S
Transfer
Blanks
To
Track S
Prog
Exit Z
N
Test
Selector
20
T
Note:
Should Not Be
Normal on this
Step
Print
&
Punch
Stop
153
Transfer
Legend "NCOSN"
Track Z To
Track Q
154
Transfer
Salesman's No.
From Core Buffer
To Track Q
155
Transfer
Salesman No.
Track F To
Track Q
Prog
Exit G
Type
Stop

THE CONTROL PANEL
FUNCTION CHART

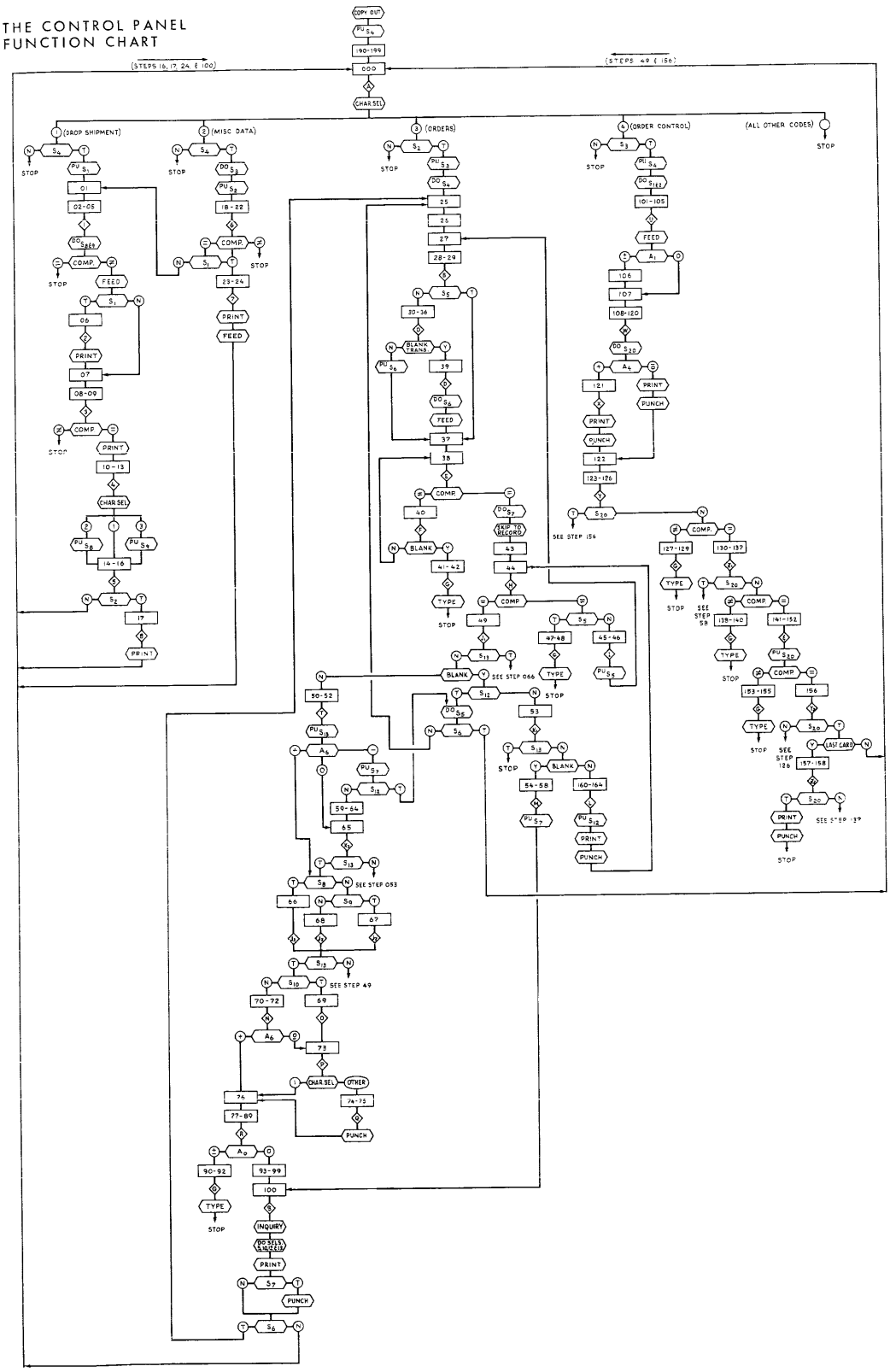
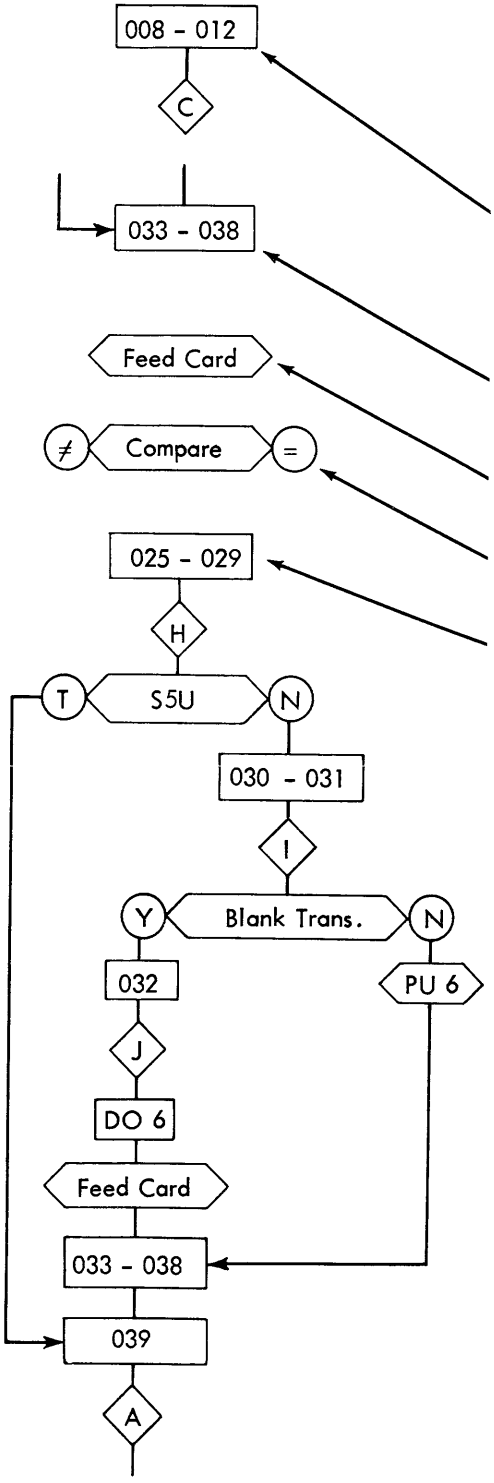


Figure 46

THE CONTROL PANEL FUNCTION CHART (Figure 46)



The chart is usually made by reference to the detailed block diagram, as follows:

One block is used for a series of program steps leading to a control exit. In this example, program steps 008 through 011 are simple transfers without control-exit codes. Step 012 contains a "C" control exit.

Lines entering a block indicate re-entry at the lowest numbered step--in this example, step number 033.

This symbol is used for function-entry hubs on the 305 control panel.

The circles indicate control-panel function exits.

This example would be interpreted as follows:

Step 025 through 028 are data transfers with normal sequential advancement.

Step 029 contains a control exit code "H" which stops the program advancement and shifts control to the 305 control panel.

Exit "H"is wired to test the upper set of points on selector 5; if transferred, re-enter to step 039; if normal, program advance to step 030.

Step 030 is a simple data transfer.

Step 031 contains control exit I, which is wired to test Blank Trans.; if "no" pick up selector 6 and re-enter at step 033; if "yes" program advance to step 032.

Step 032 contains control exit J, which is wired to drop out selector 6, feed card, and program advance to step 033.

Steps 033-038 are data transfers.

Step 039 is a data transfer with a control exit A.

An arbitrary step number assignment is made initially, and the control panel function chart is made using these numbers. The following rules should be followed when making the chart:

1. Keep chart small and on one piece of paper, if possible, so entire program can be visualized easily.

2. Segregate major routines so they are clearly separate.

3. Build sequence of blocks in a straight line down the paper.

4. Keep lines straight and cross-overs to a minimum.

5. Use standard symbols so others can interpret the meaning of the chart.

Several re-arrangements and re-drawings may be necessary to develop the most useful and clear chart.

After the chart is completed, it can be used:

1. As a check to see that all programs follow the correct path and that there are no "loose ends."

2. To present a clearer picture of the over-all logic of the program. This may show possibilities for improvement.

3. To enable others to develop quicker understanding of programs when assistance is required, especially the Customer Engineer and Sales Assistance personnel when debugging programs.

4. To develop best wiring conditions on 305 control panel to:
    a. make best use of control panel features.
    b. determine distributor requirements.
    c. re-assign program steps for minimum wiring.
    d. eliminate control exits used only to connect a broken series of program steps (by program step re-assignment).

IBM 305 RECORD LAYOUT FORM

APPLICATION DISK FILE STORED RECORDS DATE

Printed in U.S.A.

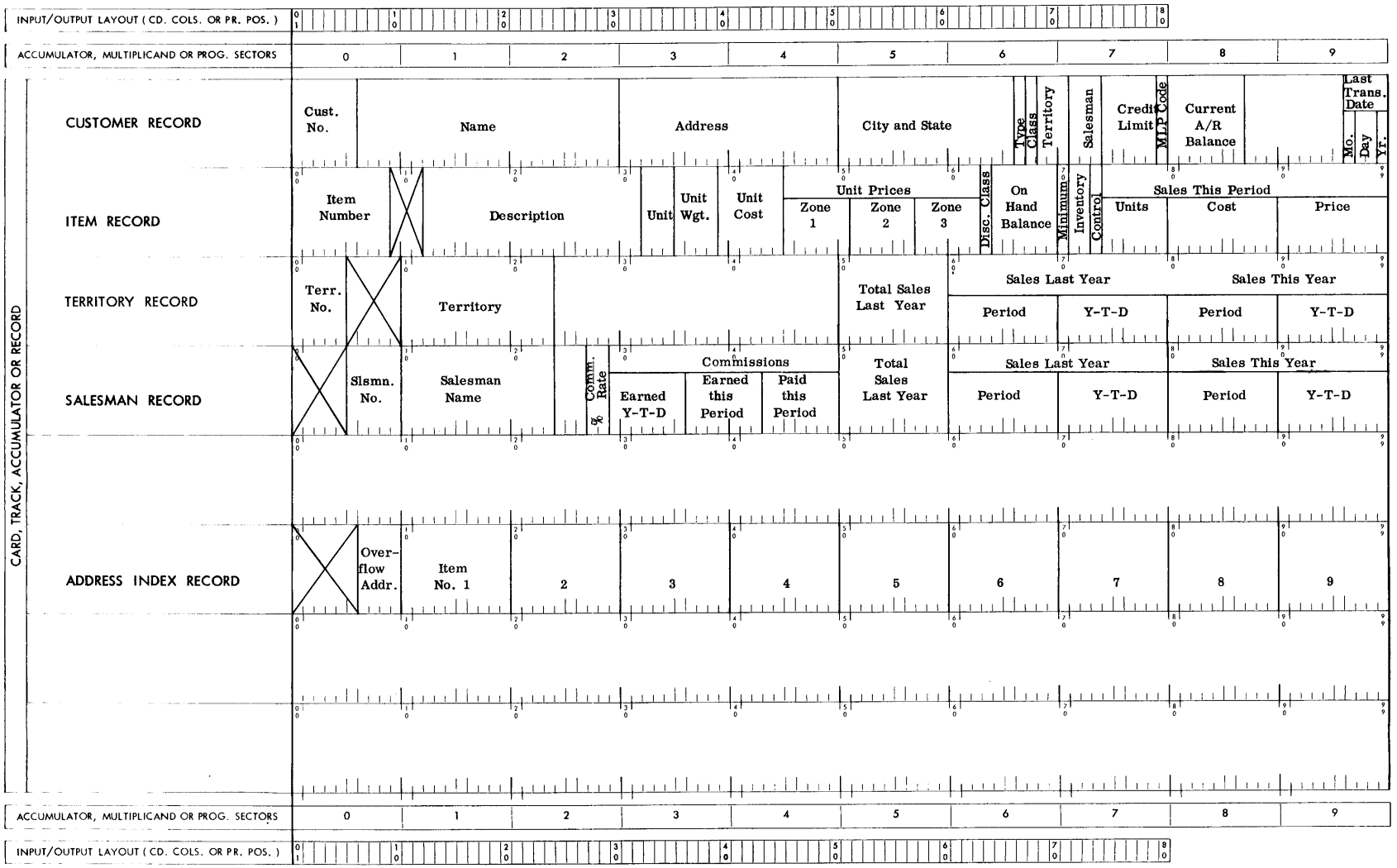


Figure 47

IBM

305 RAMAC® PROGRAM INSTRUCTION SHEET

Printed in U.S.A.

Application: EXAMPLE Date: _____ Page _____ of _____

Routine: _____ Written by: _____

PROGRAM LOADING INSTRUCTIONS

ONE				TWO				THREE			
1 FROM	4 TO	7 NO. CHAR.	9 10 CONTROL	11 FROM	14 TO	17 NO. CHAR.	19 20 CONTROL	21 FROM	24 TO	27 NO. CHAR.	29 30 CONTROL

PROG. STEP	FROM TR.	FROM POS.	TO TR.	TO POS.	NO. CHAR.	CONTROL	DESCRIPTION	TO PROG. STEP
001	W	05	S	04	05	1	Compare customer number on track W with that on Track S	
Restart: Reset and Restart on Step 001								
							Check Reset Program Start ☐	
002	W	05	J	99	05		Seek customer record	
Restart: Reset and Restart on Step 002								
							Check Reset Program Start ☐	
003	W	99	Y	98	99		Transfer contents of Track W to Track Y	
Restart: Reset and Restart on Step 003								
							Check Reset Program Start ☐	
004		99	L	99	00	5	Reset all Accumulators	
Restart:								
							Check Reset Program Start ☐	
005	K	99	W	99	00	1	Transfer Data on input track to Track W.	
Restart: Reset and Restart on Step 005							(1) Drop-out Selectors 8 and 9	
* Indicates cards out of sequence							(2) Test Compare - Equal - Stop *	
Correct and Restart on Step 000							Unequal - Test Selector 1	
							Normal ----	006
							Transferred ----	007
							Check Reset Program Start ☐	

Figure 48

PROCESS CONTROL PANEL
(IBM 305 RAMAC)

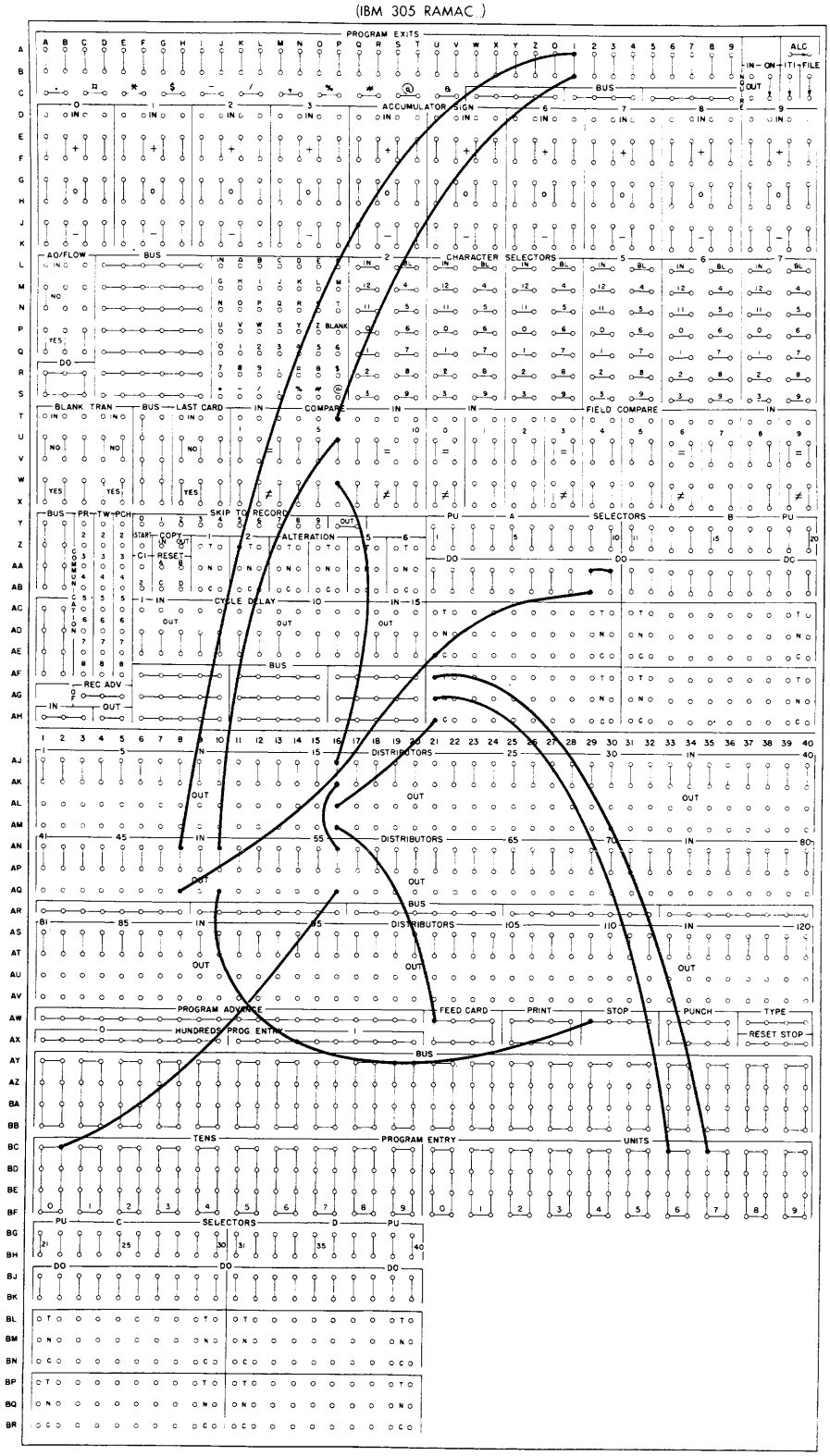


Figure 49

IBM

305 RAMAC SELECTOR ASSIGNMENT CHART

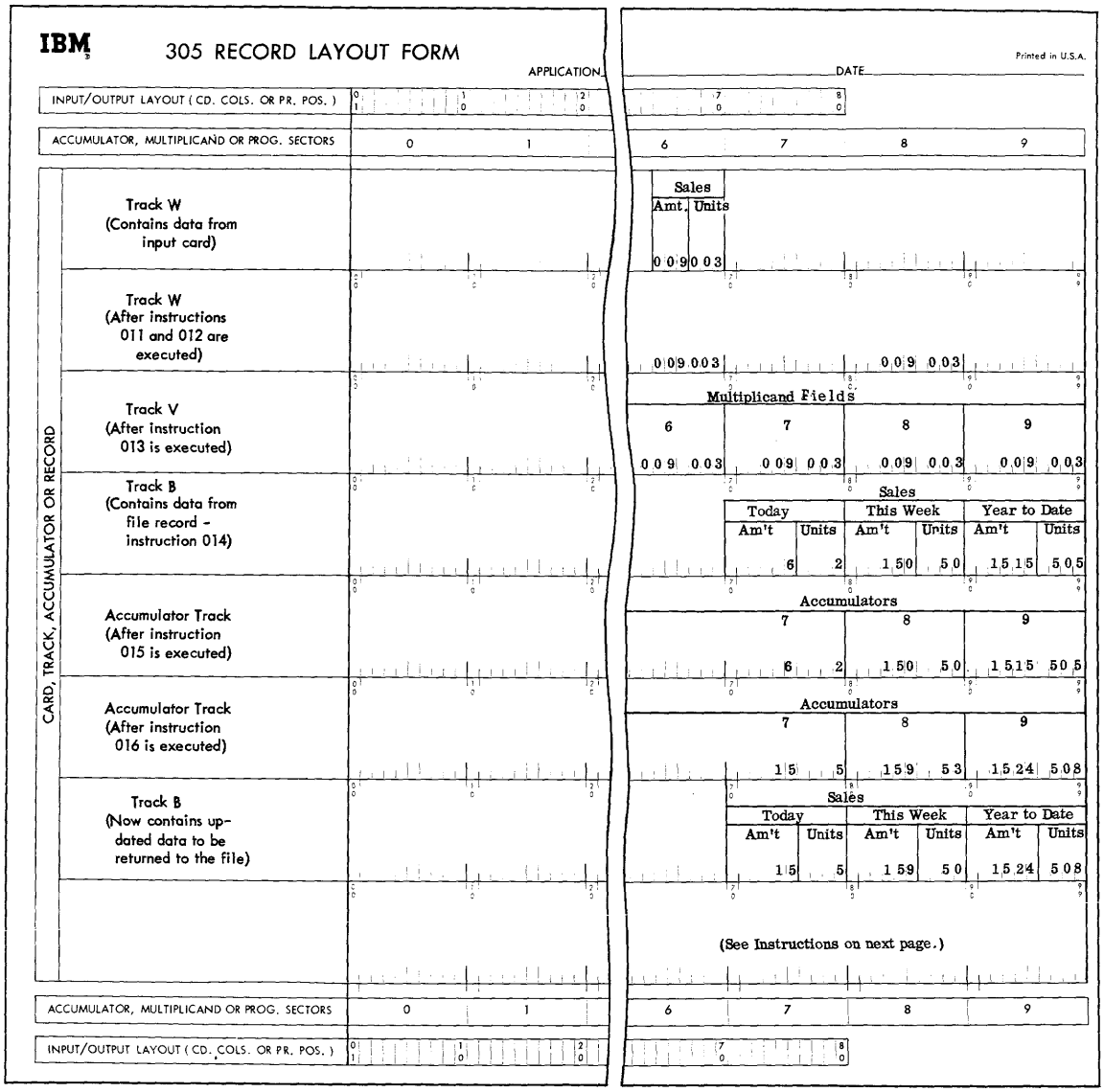
Printed in U.S.A.

Application: _____ Page: _____ of _____

Written by: _____ Date: _____

SEL. NO.	PROG. NO.	PROG. EXIT	FUNCTION	PURPOSE OF SELECTOR
1	00	A	PU: Digit 1 through N/T of Sel. 4	Transferred whenever a Type 1 (Drop Ship Address) card has been present
	00	A	DO: Digit 4 through N/T of Sel. 3	
			T: To Step 06	
			N: To Step 07	
	05	1	C:	
			T: To Step 23	
			N: To Step 01	
	22	6	C:	
2	00	A	PU: Digit 2 through N/T of Sel. 4	Transferred whenever a Type 2 (Misc. Data) card has been present. Controls printing when a Type 1 card has not been included. Checks card sequence that a No. 2 card precedes a No. 3 card.
	00	A	DO: Digit 4 through N/T of Sel. 3	
			T: PU S_3, DO S_4, go to Step 25	
			N: Stop (cards out of sequence)	
	00	A	C: Digit 3	
			T: To Step 17	
			N: To Step 000	
	16	5	C:	
3	00	A	PU: Digit 3 through N/T of Sel. 2	Controls sequence of input cards. No. 4 card must precede a No. 3 card.
	00	A	DO: Digit 2 through N/T of Sel. 4	
			T: PU S4, DO 1 and 2, Go to Step 101	
			N: Stop (cards out of sequence)	
	00	A	C: Digit 4	
			T:	
			N: (Unused)	
			C:	
4	00	A	PU: Digit 4 through N/T of Sel. 3	Controls sequence of input cards. No. 4 card must precede a No. 1 or No. 2 card.
	00	A	DO: Digit 3 through N/T of Sel. 2	
			T: PU S1. To Step 01	
			N: Stop (cards out of sequence)	
	00	A	C: Digit 1	
			T: DO3, PU S2, Go to Step 18	
			N: Stop (cards out of sequence)	
	00	A	C: Digit 2	
5	46	1	PU:	Controls automatic re-randomize and re-seek when file record fails to compare with input card.
	49 52	J T	DO: 2 conditions both through Sel. 12 N/T	
			T: Go to Step 37	
			N: Go to Step 30	
	29	B	C:	
			T: Go to Step 47	
			N: Go to Step 45	
	44	H	C: From # compare	

Figure 50



Prog. Step	FROM			TO			No. Characters		Control	
	Track	Position		Track	Position					
011	W	6	9	W	8	9	0	3		
012	W	6	6	W	8	5	0	3		
013	W	8	9	V	9	9	0	7		
014	R	9	9	B	9	9	0	0		
015	B	9	9	L	9	9	3	0		5
016	V	9	9	L	9	9	3	0		
017	L	9	9	B	9	9	3	0		

Figure 51

SPECIAL FEATURES

The following are special features that are available for the 305 RAMAC.

Additional Disk Storage

One additional IBM 350 Disk Storage Unit may be attached to the IBM 305 RAMAC to increase storage capacity from 50,000 to 100,000 (100-character) records. The 50 additional disks, which are housed in a separate frame similar to the present 350 Disk Storage Unit, use addresses from 50000 to 99999. One common address register is used for both files. Programming remains the same as with a single disk storage unit (see 305 RAMAC M Bulletin, Form 328-0768).

Dual Access

One additional access mechanism is available for each IBM 350 Disk Storage Unit attached to the 305 RAMAC system. With the addition of the second access arm it is possible to have one access unit in position for reading or writing on a record while the other unit is moving to the next record. Dual access is particularly advantageous in applications where processing has not been continuous because access time for an item exceeds the amount of processing time required (see 305 RAMAC M Bulletin, Form 22-7505).

Processing Drum Tracks

Four additional processing drum tracks are available for the 305 RAMAC. These four additional tracks: U, /, ., #, function in the same manner as the standard working storage tracks W, X, Y, and Z. See 305 RAMAC M Bulletin, Form 22-7506.

Program Exit Split

The double program exit hubs can be split and placed under selector control so that either the upper hub or the lower hub (but not both) will emit the corresponding program exit impulse. This, in effect, doubles the number of program exits available on the process control panel. See 305 RAMAC M Bulletin, Form 22-7506.

Automatic Division

With automatic division the 305 Processing Unit can divide much more rapidly than when programmed division is used. Maximum length of the divisor is nine digits, of the dividend 19 digits, and a quotient of up to 19 digits may be calculated. However, the sum of the number of digits in the divisor and quotient must not exceed twenty digits. See 305 Reference Manual, Form A26-3502-0.

Dual Process

Two IBM 305 RAMAC systems may be attached to the same 350 Disk Storage Unit(s) to form a completely new system configuration. A RAMAC with dual system control is essentially two complete RAMAC systems, except that both systems share the same disk storage unit rather than each operating with its own individual file. The two systems are independently controlled and their only connection is through the shared 350 Disk Storage Unit. See 305 RAMAC M Bulletin, Form G26-3500.

IBM 382 Paper Tape Reader

The IBM 382 Paper Tape Reader is available as an alternate input unit for the RAMAC system. Tape is read at the rate of 20 characters per second and is written on the tape input track in the same sequence in which it is read. However, rearranging the order of data on the input track can be accomplished with a special skip control feature. Paper tape input can be used separately or in conjunction with card input. See 305 RAMAC M Bulletin, Form G26-3501.

IBM 381 Remote Printing Station

As many as four remote printing stations may be attached to each IBM 305 RAMAC to provide typing facilities. Each station may be located as far as 40 feet from the 305 system; however, a customer may install and maintain cable 2,500 feet in length to extend the location of the station(s). See 305 RAMAC M Bulletin, Form G26-3503.

Input Rearrangement and Input Analysis Features

Input rearrangement and input analysis are companion features available for the IBM 305 RAMAC. The purpose of these features is to save program steps

formerly required to organize data after entry into the system and to analyze card codes. See 305 RAMAC M Bulletin, Form G26-3504.

IBM 407, Model R1 and R2

An IBM 407 Accounting Machine may be attached to the IBM 305 RAMAC. Ability to use the 407 on-line with the RAMAC not only increases the printing ability of the RAMAC system but also provides additional accounting and document accuracy controls through on-line use of 407 counter capacity.

The 305-407 may be used either on-line as a RAMAC printer-accounting machine or off-line as a normal 407. With a minimum of selection, the same control panels may function either on- or off-line. The 305-407 is available as a Model R1 or Model R2, equivalents of the 407 A1 and A2 in capacity. See 305 RAMAC M Bulletin, Form G26-3505.

APPENDIX

1. Assembly Programs and Symbolic Programming

The process of converting the English language instructions, as described on pages 27 to 29, to the coded language useful to the 305 RAMAC, may occasionally become somewhat involved. It necessitates reference to all documents prepared by the programmer, including the detailed flow charts, track layouts, and record layouts. This method of coding may sometimes produce clerical errors. Usually, the manually-coded program is not relocatable; that is, it cannot be executed from a different section of the processing unit without being manually changed. Another difficulty arises when instructions are inserted or deleted after a program has been coded. In that case, all instructions that refer to other instructions must be changed appropriately.

Symbolic programming is the writing of program steps in a language similar to that used in the flow chart. Instead of using the low-order character position and number of characters to designate a field, a name or symbol is used. The only difference is that the wording used is somewhat restrictive. Program steps can be assigned names and referred to as such. There is usually a mnemonic relationship between the name given the information and its use. Thus, it is easy to tell what data is being acted upon. Because there is only a relative relationship among symbolic instructions, routines may be relocated and program steps inserted or deleted.

The assembly program enables the system to translate a symbolic program into machine language, and may reduce the clerical effort associated with coding a program manually. If, after a program is coded, it is modified in any way (steps modified, relocated, inserted or deleted), it can be recoded by reassembling. The input consists of cards containing symbolic program steps. The output consists of a listing of the symbolic program steps and their translation. The programmer may also have a description of the program exits appear on the output sheet. This document, prepared by the RAMAC, contains all of the information normally exhibited in the detail flow chart (see pages 74-80). A deck of self-loading, one-instruction-per-card load cards is punched out.

2. 305 Trace Programs

There are several methods by which a stored program and its associated process control panel can be tested to determine whether they are correct. One is to let the program run at normal operating speed and check to see that final results are correct for the input data. A second method is to step the machine through the program one instruction at a time and check the results at various points in the program. A third method combines the advantages of the

first two. That is, it allows the same amount of detail to be obtained as with the single-step hand checkout method, and still allows the program to be traced at normal operating speeds. This third method is known as "tracing." Its purpose is to interpret the program being debugged, one instruction at a time, and then to record the results in a readable form.

For the 305 RAMAC, two trace programs have been prepared. One program traces the stored program portion. The logic is artifically imposed by the programmer when he indicates the program steps that are to be performed as one block of instructions and those which are to belong to the next block of instructions. For example, if the data being considered would normally cause program steps 010 - 015 to be performed, and then steps 041 - 053, etc., the programmer would supply these step numbers to the trace program. These steps would then be traced in the order indicated at the rate of 650 instructions per hour. Thus, the logic has been supplied without being performed through the program's control panel. This means that the stored program portion of the machine program can be traced and checked without ever wiring the control panel that would normally go with the program.

The second program allows the control panel to be traced by the RAMAC, independent of the stored program. This is accomplished by preparing a list of the program exits that are to be wired. For each program exit, the conditions of the logical elements are indicated. For example, accumulator 1 may be minus, a Z in the character selector, and the compare unit at "equal." These would be recorded on a form with the program exit and the condition of all other logical elements. This is done for every set of conditions for every program exit to be traced. The board trace uses this data to set up the conditions and actually performs the program exit. The program step number to which control is sent by the control panel is recorded by the trace program. When all program exits have been tested; the program exit, its setup conditions, the return point, and other pertinent information are printed by the RAMAC. Thus, an entire control panel can be checked out at the rate of three to four program conditions per minute.

3. Address Conversion Techniques

The following are a few of the simpler techniques used in indirect addressing:

a. Divide the control data by the range of record locations assigned, and use the remainder as the disk file address.

Example: To form a random address for file item 967588993 where 5,000 records are assigned to locations 35000 - 39999:

Modify range to 5001 (The range must be modified to an odd number.)
Divide: 9675889993 ÷ 5001 = 1934791
Remainder: 00202
Add: 35000
Disk Address: 35202

Using this method, item 9675889993 would be assigned file address 35202. The advantage of the above method lies in the fact that every digit in the control data contributes to the remainder and the remainder is properly scaled. In cases where this method has been employed, the resulting addresses have been randomly distributed.

b. Cross Indexing
This is done by punching the RAMAC address directly into the transaction file from a central index.

c. Fold the Control Data
Split the identification number, and add the two parts. Repeat this operation until the sum is a four-digit number that can be used as a disk file address.

Example: To derive a random address for file item 351134186, split the number into two parts:

351134186 = 35113 4186

Add the parts:

35113 + 4186 = 39299

Repeat the operation:

39299 = 3 + 9299
3 + 9299 = 9302
- 5
4302

Thus, 4302 becomes the file address for item 351134186.

d. Fold the end digits into the middle digits of the control data. Select the first 1/4 of the digits of the identification number and the last 1/4 of the digits. Add the first 1/4 of the digits and the last 1/4 of the digits to the middle half of the identification number. If necessary, this operation, also, may be repeated to give a number suitable for use as an address.

Example: To derive a random address for file item 21642186, select the first 1/4 and the last 1/4 of the digits:

21642186 = 21 + 6421 + 86

Add these selected digits to the middle of the identification number in this manner:

```
 6421
   86
+ 21
------
 8607
```

e. Compute the product of the halves of the control data. Split the identification number, and select the middle digits from the product of the two parts to form a random address.

Example: To derive a random address for file item 31745902, split the number:

31745902 - 3174 = 5902

Compute the products of the parts:

```
    5902
x   3174
--------
   23608
  41314
   5902
 17706
--------
18732948
- 5
--------
  2329
```

Using this method of assigning random addresses, file item 31745902 would be assigned address 2329.

f. Multiply the control data by a relatively large prime number. Multiply the identification code by a prime number; for example, 31, 41, 43. Select the high-order digits of the product to form a random address.

Example: To derive a random address for file item 317415, multiply the number by 43:

```
  317415
    x 43
  952245
1269660
--------
13648845
```

Select the high-order digits in the product to form a random address. When this method of assigned random addresses is used, file item 317415 would be assigned address 1364.

g. Multiply the control data by 1991991991. Select digits of the product to form a random address.

Example: To derive a random address for file item 317415, multiply by 1991991991:

```
        317415
 x 1991991991
--------------
  633288140523265
```

Select a segment of the product to form a random address. Using this method, file item 317415 will be assigned address 4052.

h. Add the odd-order digits in the control data to the even-order digits. Add the odd digits to the even digits. Repeat the operation on the resulting sum, if necessary, to form a random address.

31745902

Add these to the odd-order digits:

```
  1492
+ 3750
------
  5242
-  5
------
  0242
```

When this method of assigning random address is used, file item 31745902 would be assigned address 0242.

i. Split control data into groups and add the groups.

Example: To derive a random address for file item 902645813, split the number into groups of three digits:

902 645 813

902
645
+ 813
2360

When this method is used, file item 902645813 would be assigned address 2360.

j. Split the control data into groups and multiply the groups. Select digits of the product to form a random address.

Example: To derive a random address for file item 902645813, split the number into groups of three digits:

902 645 813

Compute the product of these groups:

902	581790
x 645	x 813
4510	1745370
3608	581790
5412	4654320
581790	472995270

Using this method, file item 902645813 would be assigned address 2995.

4. Load Rating Chart.

All control panel exits and distributors are designed to withstand a specific amount of electrical load which, if exceeded, may cause damage and improper operation of the machine. Therefore, when a control panel exit or distributor exit is wired to more than one control panel function, it must be determined that the electrical load is within established limits.

All control panel exits may be loaded to a maximum rating of 60 with the following exceptions:

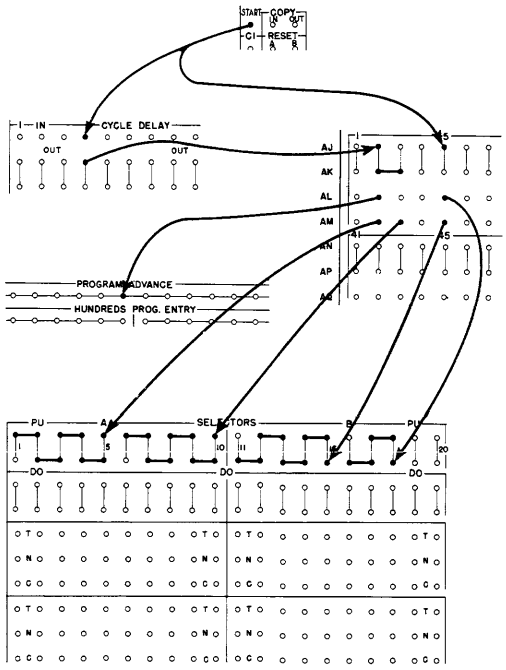
a. 305--Distributors--30 max.
--Inquiry Out --30 max.
--Start --30 max.

b. 370--Distributors--30 max.
--MLP I Exit --30 max.

c. 380--Distributors--30 max.

To insure that electrical limits are not being exceeded, reference can be made to the Load Rating Table shown in Figure 53. Add the individual load ratings for each function associated with an exit to be sure the maximum allowable load is not exceeded. Also, be sure that each distributor exit is not overloaded. Figure 52 illustrates both correct and incorrect wiring.

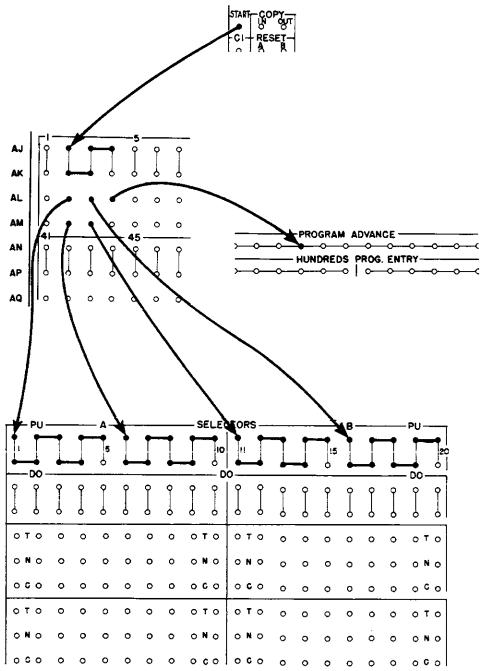
CORRECT



By using a Cycle Delay, the Start impulse is supplemented in order to pickup ten additional selectors and Program Advance.

Start	30
Cycle Delay OUT	32
Maximum Load for any Distributor Exit	30

INCORRECT



Although the loading for each distributor exit is within limits (5 selectors at a rating of 3 = 15), the total load for the Start hub (maximum 30) is 62.

Figure 52

CONTROL PANEL ENTRY HUB —305—	LOAD RATING
Accumulator Drop Out	3
Accumulator Overflow IN	0+
Accumulator Sign IN	0+
Blank Transmission IN	0+
Char Sel Alpha IN	0+
Char Sel Numeric IN	0+
Comm Channel (Print)	0+
Comm Channel (Punch)	0+
Compare IN	0+
Cycle Delay	6
Distributors	0+
Feed Card	3
Field Compare IN	0+
Hundreds Program Entry	1
Inquire IN	5
Last Card IN	0+
Print	5
Program Advance	2
Punch	3
Record Advance IN	5
Reset	3
Reset Stop	3
Selector PU	3
Selector DO	3
Skip to Record	6
Stop	1
Tens Program Entry	1
Type	5
Units Program Entry	2

CONTROL PANEL ENTRY HUB —370—	LOAD RATING
Comm Channels	0+
Co-Selector PU	2
Distributors	0+
Line End	8
Line Prog Sel PU	3
Line Space	8
MLP Start 1	2
MLP Start 2	3
MLP Start 3	4
MLP Start 4	6
Output Track	1
Print Space	1
Print Start	2
Print Stop	12
Skip to Hubs	3
Symbols	12
X Eliminate	2
NX Eliminate	2
Zero Supp Start	3
Zero Supp Stop	3

CONTROL PANEL ENTRY HUB —380—	LOAD RATING
Carriage Return	2
Clear	2
Col Ctrl. Entry	2
Col Ctrl. Delay	2
Col Ctrl. On	2
Column Split	4
Digit Selector PU	2
Distributors	0+
Program Entry	2
Program On	2
Ribbon Shift B	2
Ribbon Shift R	2
Selector PU	3
Selector DO	3
Space	8
Tab	2
Type 100	2
Type only	4
Zero Suppress Off	2
Zero Suppress On	3

NOTES:

1. When a load rating of 0+ is shown, the load is determined by the final use of the function. For example, assume a panel impulse is wired to accumulator overflow IN, overflow NO is wired to Program Advance, and YES is wired to Type and Units and Tens Entry. The load rating for the NO condition is 0+2=2. For YES, 0+5+2+1=8.

Figure 53